Worksheet 3: Recursion

Template file:	Worksheet3.hs
Labs:	Friday 22/2/2010 and 1/3/2019
Hand-in:	18.00 hr on 10/3/2019
Topics:	Recursion on natural numbers and lists. Sort.

Questions are respectively 10, 15,10,15 and 50 points.
You code should compile without errors. Test you answers.
Consult the slides before searching the web.
Don't copy from other people: you should be able to explain your own code.

1. Write a function `skipall` that given a natural number `n` (greater than 0) and a string removes every `n`-th element in the string. For instance,

   (a) `skipall 2 "abcde"` should be equal to `"ace"`.

   (b) `skipall 4 "abcde"` should be equal to `"abce"`.

2. Implement the higher order insertion sort algorithm `hoMergeSort` which is similar to merge sort except that an element `x` is placed before an element `y` if `fun x < fun y` where `fun :: a -> b` is a function taken as input by higher order merge sort. In other words, the result of evaluating `hoMergeSort fun xs` should be a list `[y1,y2,..., yn]` such that

   `fun y1 < fun y2 < ...  < fun yn`.

   Compare the type of `hoMergeSort` with the type of the function `qsortCp` given in Lecture 9.

3. Suppose we have a spreadsheet `:: [(String,String,Int)]` containing triples (name, user,mark) where name is the lastname of a student, user is his username and mark is the result of the student's first CW.

   Write a function `sortLastname` that sorts a spreadsheet lexicographically by names using the higher order insertion sort defined in Exercise 2.

   Write a function `sortUsername` that sorts a spreadsheet lexicographically by username using the higher order insertion sort defined in Exercise 2.

   Write a function `sortMark` that sorts a spreadsheet by marks in DESCENDING order using the higher order insertion sort defined in Exercise 2.

   For example

```
sortMark[("Socrates","ps21",60),("Xantippe","x12",80 ),("Cleo","cl123",70)]
```

should give the output

```
sortMark[("Xantippe","x12",80 ),("Cleo","cl123",70),("Socrates","ps21",60)]
```

4. In this question we will define a function for sorting lists based upon the algorithm `bucketsort:: Ord a => [a] -> [a]`, which can be used to sort list of elements of any type as long as that type is provided with an order relation.

   - First, define a function `smallest` which takes as input a list and returns the smallest element in the list
   - Next, define a function `delete` which takes as input an element and a list and returns the list obtained by deleting the first occurrence of the element in the list
   - Finally define the function `bucketsort` which takes a list and returns the list whose head is the smallest element in the list and whose tail is the result of recursively sorting the list obtained by deleting the smallest element of the list from the list.

5. We want to print chessboards of various sizes. For instance a 3x3 board:

```
.----------------.
|    ****    ****|
|    ****    ****|
|****    ****    |
|****    ****    |
|    ****    ****|
|    ****    ****|
|****    ****    |
|****    ****    |
.----------------.
```

   We think of a chessboard as made from tiles. We will represent tiles as lists of strings usually of the same length. So let `Tile = [String]` and `Board = [[Tile]]`. We think of elements of `Board` as a list of rows (lists) of tiles. The tile `["****","****"]` will be printed like:

```
****
****
```

   by printing the string "$\backslash n * * * * \backslash n * * * * \backslash n$" using `putStr`.

(a) Write a function `makeTile :: Char -> Int -> Tile` that given a character `c` and an integer `n` produces a a tile consisting of `n/2` lines of `n` characters `c`. For example: `makeTile '*' 4 = ["****","****"]` and `makeTile '*' 5 =["*****","*****"]`

(b) In order to print a tile we must be able to transform a tile like `["****","****"]` into a string `"\n****\n****\n"`. Write a function `tile2string :: [String] -> String` that takes a tile, that is, takes a list of strings and transforms it into one long string as in the example, such that the function

```
printTile :: Tile -> IO()
printTile tile = putStr(tile2string (tile))
```

nicely prints your tiles. (In fact this is rather similar to the display function of the previous exercise. Can you see a polymorphic definition that will work in both cases?)

(c) To print boards we have to glue tiles horizontally as well as vertically as suggested by the following

```
&&&                  &&&***
&&&                  &&&*** (horizontal glueing)
***
*** (vertical glueing)
```

Hence write a function `vglue :: Tile -> Tile -> Tile` that glues two tiles vertically

(d) Next write a function `hglue :: Tile -> Tile -> Tile` that glues two tiles horizontally.

(e) Using all this we can now try to print boards: the idea is that we glue the tiles of one board together so that we get one big tile that we now know how to print. Before we will build this gluing function `board2tile :: Board ->Tile` we make two help functions.

Write a function `row2tile :: [Tile]->Tile` that will glue a row of tiles horizontally to one long tile.

Write a function `col2tile :: [Tile]->Tile` that will glue a column of tiles vertically to one tall tile.

(f) So, if we think of a board as a column of rows of tiles, then we can convert a board into a tile using `col2tile` and `row2tile` Now write a function `board2tile :: Board ->Tile` that transforms a board of tiles into one big tile:

For instance

```
board2tile [[["**"],["  "],["**"]],[["  "],["**"],["  "]],[["**"],["  "],["**"]]
=
["**  **","  **  ","**  **"]
```

(g) We now want to print an edge around a tile with a function `edge :: Tile -> Tile`. For example `edge` should transform the tile `[["****","****"]]`

in the tile `[".----.","|****|","|****|",".----."]`, which will print as:

```
.----.
|****|
|****|
.----.
```

Using this function we can now also print boards with an edge, provided we first glue them into a big tile.

(h) (Perhaps hard) Now we are ready to make and print chessboards. See the start of this question for an example of a three by three chessboard: we will make the white tiles using `makeTile ' ' n` and black tiles using `makeTile '*' n`. Write a function `chessboard :: Int -> Board` that produces a chessboard with a **black square in the bottom left.**

For example `chessboard 8` can now be printed with an edge resulting in the following 8x8 chessboard as output: (see next page)