Functional and Logic Programming

Home Assignment 2

Due: Saturday, 28.4.2018 - 23:55

Instructions

- Please create a source file called ***HW2.hs*** and put all the answers there.
  The file should start with a comment which contains your **full name** (in English) and **ID**
  (see also example: ***hwexample.hs*** in the Moodle)

  ```
  -- John Doe
  -- 654321987
  ```

- Make sure the file **is valid** by loading it into GHCi.
  A valid file will load without any errors or warnings.

- If you need a function but you don't know how to implement it - just write it's signature (name and type) and put `undefined` in the function's body.
  That way you'll be able to load the file even though it contains references to undefined names.
  (see also example: ***hwexample.hs*** in the Moodle)

- When writing a function - write both the **type** and the **body** of the function.

- Be sure to write functions with **exactly the specified name** (and **type signature** - if it is provided) for each exercise.
  You may create additional auxiliary/helper functions with whatever names and type signatures you wish.

- Try to write **small functions** which perform just **a single task**, and then **combine** them to create more complex functions.

Exercises

1. **a)** Implement the function **`replaceElement`** which takes a pair `(i,x)` (where `i` is an index of type `Int`, and `x` is a **polymorphic** item), and a **polymorphic** list - and sets the `i`'th element of the list to be `x`.
If `i` is negative or bigger than the length of the list - it just returns the list without changing it.

```
replaceElement (3,'x')  "Hello"   = "Helxo"
replaceElement (-5,'x') "Hello"   = "Hello"
replaceElement (0,5)    [1,2,3,4] = [5,2,3,4]
replaceElement (100,5)  [1,2,3,4] = [1,2,3,4]
replaceElement (0,5)    []        = []
```

b) Implement the function **`replaceElements`** which takes a list of pairs of `(i,x)` (where `i` is an `Int`), and a list `xs`, and for each pair `(i,x)` it sets the `i`'th element in `xs` to be `x`.
If a pair contains in its 1st component a negative index or an index bigger than the length of `xs` - ignore this pair.
If the pairs list contains more than one pairs with the same key (i.e - `[(1,'a'),(1,'b')]`) - the **last** occurrence of the key will be the one that holds (see example).

```
replaceElements [(1,'a'), (-4,'t'), (3,'b')] "#####" = "#a#b#"
replaceElements [(2,50), (50,2)] [8,7,6,5,4] = [8,7,50,5,4]
replaceElements [(3,'a'),(5,'b'),(3,'c')] "wwwwwww" = "wwwcwbw"
```

2. In this section we'll use a list of pairs to represent a simplified "database" where each item has a **non-unique** "key" (which is given as a `String`).
The database will support the following operations:

- **`addItem`** - adds a (key,item) pair to the database.
- **`subsetByKey`** - get all the items with the given key.
- **`subsetByKeys`** - get all the items with one of the given keys.
- **`getKeys`** - get a list of all the keys.
- **`groupByKeys`** - group all items by their keys.

The database should be **polymorphic** - the implemented functions should work for **any** type of items (while the type of keys **must** be `String`).

a) Implement the **`addItem`** function which takes a pair of `(key,item)` (where `key` is a `String` and `item` is **polymorphic**), and a list of `(key,item)` pairs - and adds the pair to beginning of the list.

```
addItem ("a",8) [("",5),("x",2),("a",12)]
    = [("a",8),("",5),("x",2),("a",12)]
```

b) Implement the **subsetByKey** function which takes a `key` (given as a `String`), and a list of `(key,item)` pairs - and returns all the items in the list, which have the given `key`.

```
subsetByKey "a" [("a",8),("",5),("x",2),("a",12)] = [8,12]
```

c) Implement the **subsetByKeys** function which takes a list `keys` of keys, and a list `xs` of `(key,item)` pairs - and returns all the items in `xs`, which have one of the keys in `keys`.
You may assume the keys in the `keys` do not repeat (so you don't have to worry about cases such as `subsetByKeys ["a", "b", "a"]`).

```
subsetByKeys ["a","x"] [("a",8),("",5),("x",2),("a",12)] = [8,12,2]
```

d) Implement the **getKeys** function which takes a list `xs` of `(key,item)` pairs and returns a list of all the **keys** of `xs`, where each key appears **only once!**.
(**note:** the **order** of the elements in the result **doesn't matter!** - as long as all the keys are in the list and there are no duplicates)

```
getKeys [("a",1),("b",2),("a",3),("a",4),("c",5),("b",6)]
    = ["a","b","c"]
```

e) Implement the **groupByKeys** function which takes a list `xs` of `(key,item)` pairs and returns a list of `(key,items)` pairs where each pair contains a key and a list of all the items in `xs` which have this key.
Each key must appear **only once** in the result.
(**note:** the **order** of the elements in the result **doesn't matter!**)

```
groupByKeys [("a",1),("b",2),("a",3),("a",4),("c",5),("b",6)]
    = [("a",[1,3,4]),("b",[2,6]),("c",[5])]
```

3. In this section we'll use a list of lists to represent a matrix, where each row in the matrix will be represented by a list.

For example the following matrix:

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

will be represented as:

```
[ [1 2 3], [4 5 6], [7 8 9] ]
```

All the functions in this section, except **addMatrices** (subsection **e**), should be **polymorphic** - they should work for **any** type of elements.

a) Implement the **createMatrix** function which takes 2 **positive** Ints - m and n, and a list xs, and creates a matrix of size $m \times n$ (m rows and n columns).
You may assume the list xs will always be of the size $m \cdot n$

```
createMatrix 2 3 [1,2,3,4,5,6] = [[1,2,3],[4,5,6]]
createMatrix 3 2 [1,2,3,4,5,6] = [[1,2],[3,4],[5,6]]
```

b) Implement the **getElementInCell** function which takes 2 Ints m and n and a matrix - and returns the element in the m'th row and n'th column of the matrix.
You may assume m and n will always be within the boundaries of the matrix.

$$\texttt{getElementInCell } 1\ 2 \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} = 6$$

c) Implement the **appendH** function which takes 2 matrices and appends them **horizontally**.
You may assume both matrices have the same number of rows.

$$\texttt{appendH} \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \begin{bmatrix} 10 & 20 & 30 \\ 40 & 50 & 60 \\ 70 & 80 & 90 \end{bmatrix} = \begin{bmatrix} 1 & 2 & 3 & 10 & 20 & 30 \\ 4 & 5 & 6 & 40 & 50 & 60 \\ 7 & 8 & 9 & 70 & 80 & 90 \end{bmatrix}$$

d) Implement the **appendV** function which takes 2 matrices and appends them **vertically**.
You may assume both matrices have the same number of columns.

$$\texttt{appendV} \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \begin{bmatrix} 10 & 20 & 30 \\ 40 & 50 & 60 \\ 70 & 80 & 90 \end{bmatrix} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \\ 10 & 20 & 30 \\ 40 & 50 & 60 \\ 70 & 80 & 90 \end{bmatrix}$$

e) Implement the **addMatrices** function which takes 2 matrices of Ints and returns the result of their matrix addition (adding each 2 corresponding values).
You may assume both matrices have the same dimensions.

$$\texttt{addMatrices} \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \begin{bmatrix} 1 & 2 & 3 \\ 40 & 50 & -6 \\ 1 & 0 & -2 \end{bmatrix} = \begin{bmatrix} 2 & 4 & 6 \\ 44 & 55 & 0 \\ 8 & 8 & 7 \end{bmatrix}$$