HW 9 **Prof. Caldwell**
Due: 27 September 2011 **COSC 3015**

1 List Comprehensions

As mentioned in class, list comprehensions provide a syntactically concise and expressive means of writing list functions and every Haskell programmer should know how to use them. You should read about them in Bird (pp. 112-115) and in LYAHFGG (Chapter 2).

As examples (in addition to the ones in the readings) *take* and *drop* can be implemented using list comprehensions as follows:

```
take' k xs = [ xs !! i | i ← [0..(k-1)]]
drop' k xs = [ xs !! j | j ← [k..(length xs - 1)]]
```

Note that *!!* is the select operator for indexing into a list, thus *(xs !! j)* returns the $(j+1)^{st}$ element (zero based indexing) of the list *xs*. *[0,1,2] !! 1 == 1* (the index is 1 but 1 is the second element in the list.)

Problem 1.1. In this assignment you will use list comprehensions to implement the functions to support operations on a toy relational database representing the personnel files of the ACME Programming Company. Tables are represented in Haskell as lists of lists.
We define some datatypes to represent the database.

```
data Position = CEO | Manager | Programmer | Intern  deriving (Eq,Show)

data Field = EmployeeId Int | T Position | Name String | Salary Int deriving (Eq)
instance Show Field where
   show (EmployeeId k) = show k
   show (T p) = show p
   show (Name s) = s
   show (Salary k) = show k

type Column = Int
type Row = [Field]
type Table = [Row]
```

Here are some tables.

```
employees =
 [[EmployeeId 1234, Name "Smith"],
  [EmployeeId 1235, Name "Jones"],
  [EmployeeId 1236, Name "Brown"],
  [EmployeeId 1237, Name "Green"],
  [EmployeeId 1238, Name "Edwards"]]
salary =
 [[EmployeeId 1234, Salary 1000000],
  [EmployeeId 1235, Salary 100000],
  [EmployeeId 1236, Salary 55000 ],
  [EmployeeId 1237, Salary 78000],
  [EmployeeId 1238,Salary 0]]
```

```
positions =
 [[EmployeeId 1234, T CEO],
  [EmployeeId 1235, T Manager],
  [EmployeeId 1236, T Programmer],
  [EmployeeId 1237, T Programmer],
  [EmployeeId 1238, T Intern]]
stock_options =
 [[Id 1234, Shares 1000000],
  [Id 1235, Shares 1000],
  [Id 1236, Shares 5000 ],
  [Id 1237, Shares 2500],
  [Id 1238, Shares 5]]
```

Using list comprehensions, you need to implement functions having the following types.

delete :: Column → Row → Row
: The delete function removes the specified column from the row. Remember to implement the function using a list comprehension.

join :: (Table,Column) → (Table,Column) → Table
: The join function performs a kind of consistent merge on two tables by matching values in the named columns. If the call is of the form *join (t1,i) (t2,j)* the resulting table is created by appending rows from $t1$ to rows of $t2$ (from which column j has been deleted) whenever the value in column i of a row from $t1$ is equal to the value in column j of a row of $t2$. Deleting column j from rows of $t2$ (before appending them with the matching rows of $t1$) guarantees that the join operation does not duplicate a copy of the column. Examining the expected output for the test cases should make the more clear. Remember, use a list comprehension to implement the function.

project :: [Column] → Table → Table
: In a call of the form *project columns t* a new table is constructed from table t containing the columns of t as listed in the argument *columns*. If the list of columns is empty, the resulting table is empty. Using a projection it is possible to change the order and number of occurrences of the columns in a table. For example, if a t table has three columns, the projection *project [2,1,0,1] t* results in a table with four columns where the first is the third column from t, the second is the second column from t, the third is the first column from t and the fourth is another copy of the second column from t. Remember, you need to use a list comprehension to implement the function.