SE-318 August 2014 Assignment 2

Set: Tuesday 9 September 2014

Due: 17:00 (5pm) Monday 15 September 2014

Marks: 15% of SE-318 total

Problem statement: rewrite the solution to Assignment 1 as required in the comments to the Assignment 1 Solution below. Only change what needs to be changed!

Supplementary Information - Assignment 1 Solution (to be modified as specified)

```haskell
data Sex = Male | Female deriving (Eq, Show)
instance Ord Sex where
    compare Male Male = EQ
    compare Male Female = LT
    compare Female Male = GT
    compare Female Female = EQ

data Status = Alive | Dead | Abdicated deriving (Eq, Show)

data Person = Person Sex Status String deriving (Eq, Show)
instance Ord Person where
    compare (Person sx1 _ _) (Person sx2 _ _) = compare sx1 sx2

data Dynasty =
    Descend Person [Dynasty] | Dull Person deriving (Eq, Show)
instance Ord Dynasty where
    compare (Descend p1 _) (Descend p2 _) = compare p1 p2
    compare (Descend p1 _) (Dull p2 ) = compare p1 p2
    compare (Dull p1 ) (Descend p2 _) = compare p1 p2
    compare (Dull p1) (Dull p2 ) = compare p1 p2

successors :: String -> Dynasty -> [String]
successors name dynasty = aliveafter name (linefrom dynasty)

-- define the catamorphism cataD on Dynasty

-- then reimplement linefrom to use cataD instead of explicit
-- recursion and the new version of reorder below
linefrom :: Dynasty -> [Person]
linefrom dy =
    case reorder dy of
        Descend (Person _ Abdicated _) ds -> []
        Descend person ds -> [person] ++ (concat . map linefrom) ds
        Dull (Person _ Abdicated _) -> []
        Dull person -> [person]

-- redefine reorder so that all sub-dynasties are sorted
-- with Males before Females, using cataD
reorder :: Dynasty -> Dynasty
reorder (Descend person ds) = Descend person (sortds ds)
reorder d@(Dull _) = d

sortds :: [Dynasty] -> [Dynasty]
-- reimplement sortds to use new insertd and flatten below
sortds dys = foldr insertd [] dys

-- define a type of binary trees for Dynasty
-- data BTD = Dnode BTD Dynasty BTD | Dnull

-- define the catamorphism cataBTD on the above type

-- use cataBTD to define a function to "flatten" btd :: BTD
-- in an in-order traversal
-- flatten :: BTD -> [Dynasty]

-- redefine insertd so that "flatten d" yields every top-level
-- Dynasty in d headed by a Male before every top-level Dynasty
-- in d headed by a Female, in particular so that
-- "flatten(insertd d btd)" yields d before every top-level Dynasty
-- headed by a Person of the same Sex as d
-- insertd :: Dynasty -> BTD -> BTD
insertd :: Dynasty -> [Dynasty] -> [Dynasty]
insertd dy [] = [dy]
insertd dy (d:ds) = if  dy > d then d:(insertd dy ds) else dy:d:ds

aliveafter :: String -> [Person] -> [String]
aliveafter name ps =
    let fromnam = dropWhile (\(Person _ _ pname)-> name /= pname) ps
    in if null fromnam then [] else alivein (tail fromnam)

alivein :: [Person] -> [String]
alivein ps =
    map
    (\(Person _ _ name) -> name)
    (filter (\(Person _ st _) -> st == Alive) ps)
```

Example Dynasty

`exdyn =` as per assignment 1

Correct Solution based on Example

```
successors "Beatrice" exdyn
```

➔

as per assignment 1

```
linefrom exdyn
```

➔

as per assignment 1

```
reorder exdyn
```

➔

```
Descend (Person Male Dead "George5")
[
      Descend (Person Male Abdicated "Edward8") [],
      Descend (Person Male Dead "George6")
      [
            Descend (Person Female Alive "Elizabeth2")
            [
                  Descend (Person Male Alive "Charles")
                  [
                        Descend (Person Male Alive "William")
                        [
                              Descend (Person Male Alive "George") []
                        ],
                        Descend (Person Male Alive "Harry") []
                  ],
                  Descend (Person Male Alive "Andrew")
                  [
                        Dull (Person Female Alive "Beatrice"),
                        Dull (Person Female Alive "Eugenie")
                  ],
                  Descend (Person Male Alive "Edward")
                  [
                        Dull (Person Male Alive "James"),
                        Dull (Person Female Alive "Louise")
                  ],
                  Descend (Person Female Alive "Anne")
                  [
                        Descend (Person Male Alive "Peter")
                        [
                              Dull (Person Female Alive "Savannah"),
                              Dull (Person Female Alive "Isla")
                        ],
                        Dull (Person Female Alive "Zarah")
                  ]
            ],
            Descend (Person Female Dead "Margaret")
            [
                  Dull (Person Male Alive "David"),
                  Dull (Person Female Alive "Sarah")
```

```
                ]
        ],
        Dull (Person Male Dead "Henry"),
        Dull (Person Male Dead "George"),
        Dull (Person Male Dead "John"),
        Dull (Person Female Dead "Mary")
]
```

where exdyn is as above. (Newlines and indentation have been added by me to aid formatting in the above outputs, but you should not put these in your answer.)

Assessment Criteria

Presentation *The program should be laid-out in a readable, appropriately indented form, with sensible choice of names for functions and parameters (other than as specified above). Comments (and explicit type specifications) should be used to describe the behaviour of any functions you define, and also where needed to describe any code that needs further explanation.*	9 marks
Correctness *The program gives the correct output for the Example above, for operations* *1. linefrom exdyn* *2. successors "Beatrice" exdyn* *3. reorder exdyn*	6 marks
Total	**15 marks**

Note that the assessment is cumulative. That is:

- if the syntax is invalid, it will not be possible to get any marks for Presentation or Correctness
- marks for presentation will be proportional to the extent to which the program works correctly i.e. multiplied by the mark for Correctness divided by 6.

Submission Intructions

You should submit Haskell source code plus evidence of testing as required above.

Send one (1) email to **TBA** by the due date and time in the precise format as follows:

- subject line: "SE-318 assignment 2" followed by your name and SYSU student number
- no body, but attachments as follows:
    1. a file named "a2.hs" – your source code, with definitions in order (from top to bottom)

- data types Sex, Status, Person and Dynasty as above
- functions as defined, modified or required in order as per the assignment 1 solution above

  Any other functions you think you need to define should be defined immediately after the function that uses them.

  Leave at least one blank line between every function definition.
- After the last function definition include the definition of exdyn from assignment 1 so that you can use the name "exdyn" in your testing to refer to the test graph.

2. a file named “a2.ss”, being a screenshot of your testing of

```
linefrom exdyn

successors "Beatrice" exdyn
```

No submissions received after the due deadline will be assessed, and instead will be given a mark of 0/15.

Paul Bailes

9.9.2014