Midterm (100 pts)

Burak Ekici

Assigned : November the 10th, 10h30
Duration : 150 minutes

Name-Surname:

1 List Manipulations (35 pts)

Q1. Write below described functions in Haskell without employing Prelude library functions and the list comprehension.

a) (**10 pts**) `rotateN :: Int -> [a] -> [a]` takes an integer `n` and a list `[a]` of arguments, and right rotates the list `n` times.

   Expected behavior of the function:

```
rotateN 2 [7,5,1,3,6]   ==   [1,3,6,7,5]
rotateN 4 [7,5,1,3,6]   ==   [6,7,5,1,3]
rotateN 5 [7,5,1,3,6]   ==   [7,5,1,3,6]
rotateN 6 [7,5,1,3,6]   ==   [5,1,3,6,7]
```

b) (**10pts**) `deleteKey :: (Eq a) => a -> [(a,b)] -> [(a,b)]` takes a `key` (of some Eq class instance type `a`) with a list of pairs `l`, and returns a list of pairs that does not contain (`k`,`v`) pairs such that `key == k`.

Expected behavior of the function:

```
deleteKey "happy" [("happy",3),("sad",2)]                    ==   [("sad",2)]
deleteKey "S" [("S",2010),("S",2006),("B",2002),("S",2014)]   ==   [("B",2002)]
deleteKEy 5 [(5,7),(5,8),(6,9),(7,12)]                        ==   [(6,9),(7,12)]
```

c) (**15pts**) `partition :: (a -> Bool) -> [a] -> ([a],[a])` takes a predicate, `p` along with a list `l`, and returns a pair of lists (`l1`,`l2`) such that `l1` contains members of `l` satisfying `p` while `l2` contains those that do not satisfy `p`. Note that in both `l1` and `l2`, elements are supposed to be in the same order with that of `l`.

Expected behavior of the function:

```
partition odd []         ==  ([],[])
partition odd [1..10]    ==  ([1,3,5,7,9],[2,4,6,8,10])
partition even [1..10]   ==  ([2,4,6,8,10],[1,3,5,7,9])
partition (> 5) [1..10]  ==  ([6,7,8,9,10],[1,2,3,4,5])
```

2 User Defined Data Types (40 pts)

Q2. Within the context of a type `BoolExp` of propositional logical formulas, and a type `BValuations` of Boolean valuations, implement below stated functions in Haskell without employing Prelude functions and list comprehension.

```
data BoolExp where
    Prop :: Char    -> BoolExp
    And  :: BoolExp -> BoolExp -> BoolExp
    Or   :: BoolExp -> BoolExp -> BoolExp
    Impl :: BoolExp -> BoolExp -> BoolExp
    Iff  :: BoolExp -> BoolExp -> BoolExp
    Not  :: BoolExp -> BoolExp

type BValuation = [(Char, Bool)]
```

Notice.

The proposition `And (Prop 'p') (Prop 'q')` denotes the logical "and" of propositional formulae p and q ($p \wedge q$) while `Impl (Prop 'p') (Prop 'q')` represents the proposition p "implies" q, namely $p \implies q$. Similarly, `Iff (Prop 'p') (Prop 'q')` stands for the "double way implication" in between p and q, $p \iff q$.

In a valuation like `[('p', True), ('q', False)]`, the proposition `Prop 'p'` takes the value of `True` while `Prop 'q'` is `False`.

a) (**10 pts**) `printBoolExp :: BoolExp -> String` represents formulae in the string form.

Expected behavior of the function:

```
printBoolExp (Prop 'p')                                ==   "p"
printBoolExp (And (Prop 'p') (Prop 'q'))               ==   "(p && q)"
printBoolExp (Or (Prop 'p') (Prop 'q'))                ==   "(p || q)"
printBoolExp (Impl (Prop 'p') (Prop 'q'))              ==   "(p -> q)"
printBoolExp (Iff (Prop 'p') (Prop 'q'))               ==   "(p <-> q)"
printBoolExp (Not (Prop 'p'))                          ==   "(~p)"
printBoolExp (And (Prop 'r') (Iff (Prop 'p') (Prop 'q')))   ==   "(r && (p <-> q))"
```

b) (**15 pts**) `propNames :: BoolExp -> [Char]` extracts proposition names out of a given formula where no name duplications allowed. The appearance order also matters.

Expected behavior of the function:

```
propNames (Prop 'p')                                                    ==  ['p']
propNames (Or 'q' (And (Prop 'p') (Prop 'q')))                          ==  ['q', 'p']
propNames (Or (Prop 'r') (And (Prop 'q') (Impl (Prop 'p') (Prop 'q')))) ==  ['r', 'q', 'p']
```

c) (**15 pts**) `beval :: BValuation -> BoolExp -> Bool` evaluates the given formula under a valuation, and returns the `Bool` value.

Expected behavior of the function:

```
beval [('p', True), ('q', False)] (Or (Prop 'p') (Prop 'q'))        ==  True
beval [('p', True), ('q', False), ('r', True)]
 (Impl (Or (Prop 'p') (Prop 'r')) (Prop 'q'))                         ==  False
beval [('p', True), ('q', False), ('r', True)]
 (Iff (Not (Or (Prop 'r') (And (Prop 'p') (Prop 'q')))) (Prop 'r'))  ==  False
```

3 Untyped λ-Calculus (25 pts)

Q3. β-reduce below untyped λ-terms, as far as possible, employing the normal (leftmost-outermost) reduction order. Clearly demonstrate every single reduction step.

a) (**10 pts**) $\big((\lambda f.\,\lambda x.\,f\,f\,f\,x)\,f\big)\big((\lambda f.\,\lambda x.\,f\,f\,f\,x)\,f\big)$

b) (**15 pts**) $\big(\lambda u.\,\lambda w.\,\lambda f.\,\lambda x.\,w\,f\,(u\,f\,x)\big)(\lambda f.\,\lambda x.\,f\,f\,f\,f\,x)\;(\lambda f.\,\lambda x.\,f\,f\,f\,f\,f\,f\,x)$