COP 4020 Programming Languages

Fall 2016

Instructor: Dr. Pawel Wocjan

Final Exam

Thursday 12/08/2016

First Name: _____

Last Name: _____

NID: _____

First Name: _____

Last Name: _____

NID: _____

Problem:	Points:	
1	_____	**(out of 10)**
2	_____	**(out of 5)**
3	_____	**(out of 5)**
Total	_____	**(out of 20)**

Problem 1 (You get 1 point for each correct answer. There are two bonus questions.)

1. Give the definition of a higher-order function.

2. Give the definition of persistence.

3. Determine the kind of `Maybe`.

4. Determine the kind of `Either`.

5. Determine the kind of the partially applied type constructor `Either String`.

6. Determine the signature of the function `filter`.

7. Determine the signature of the function `map`.

8. Determine the signature of the lambda function `(\x -> x:[])`.

9. Determine the signature of the function `return`.

```
return ::
```

10. Determine the signature of the bind operator `(>>=)`.

```
(>>=) ::
```

11. Give two examples of built-in classes in Haskell.

12. Give the desugared version of the list `[1,2,3]`.

Problem 2

You are given the algebraic data type

```
data KaryTree a = Node a [KaryTree a] deriving (Show,Eq)
```

that models non-empty k-ary trees, in which a node contains an element and has an arbitrary number of subtrees. For instance:

would be represented as

```
Node 1 [Node 2 [],Node 3 [Node 5 [],Node 6 [Node 7 []]],Node 4 []]
```

Implement the function **`count`**

```
count :: (a -> Bool) -> KaryTree a -> Integer
```

that takes as first input a predicate and as second input a tree, and outputs how many elements stored in the tree satisfy that predicate. For instance:

`count odd t ~~> 4` and `count even t ~~> 3` where `t` is the above tree.

You may assume that you have the function `bool2Integer :: Bool -> Integer` such that

`bool2Integer False ~~> 0` and `bool2Integer True ~~> 1`

```
data KaryTree a = Node a [KaryTree a] deriving (Show,Eq)
count :: (a -> Bool) -> KaryTree a -> Integer
```

Problem 3

Implement the function `suffixes` that takes a list as input and produces as output the list that contains all possible suffixes of the input list. For this problem the empty string is not considered to be a suffix. For instance:

```
suffixes "smart" ~~> ["smart","mart","art","rt","t"]

suffixes [1,2,3] ~~> [[1,2,3],[2,3],[3]]
```

Your implementation has to be polymorphic and you have to include its signature.