Programming Exercise 3

Submit Assignment

Due Monday by 11:59pm **Points** 50 **Submitting** a file upload

Haskell Programming

Due Monday, April 16

1. Create the function removedups that takes a list and removes duplicate elements.

```
removedups [1,2,2,3,3,3,4,3,4,5,5,5,4,3,3,2,1] => [1,2,3,4,3,4,5,4,3,2,1]
```

2. Create a continuation passing version, removedups-cps

```
removedups_cps [1,2,2,3,3,3,4,3,4,5,5,5,4,3,3,2,1] (\v -> v) => [1,2,3,4,3,4,5,4,3,2,1]
```

3. While Haskell is similar to Scheme, Haskell's type rules prevent us from writing a function like the *-functions of the first Scheme homework. For example, we can't write the equivalent of (removedups* '(1 1 1 3 3 (4 ((5 5) ())) 6 6 6)) because a list can't contain both int types and list types as elements. You will fix this by creating the following type.

Create a type that allows us to have nested lists. Your type should have two kinds of values, elements and sublists. For example, the following will be a valid list:

```
[Element 1,Element 3,Sublist [Element 4,Sublist [Sublist [Element 5],Sublist []]],Element 6]
```

4. Create the function gremovedups that takes a list containing elements and sublists and returns a list with the same structure, but if any "element" is preceded by an identical element, that element is removed.

```
gremovedups [Element 4,Element 4,Element 5,Sublist [Element 6,Element 6,Sublist[Element 8,Element 8,
Element 8]], Element 5,Element 5] =>
[Element 4,Element 5,Sublist [Element 6,Sublist [Element 8]],Element 5]
```

5. Using the Tree type created in class (use a version without lists), write a function bubbledown that takes a Tree as input. If the element stored in the root is larger than either children, swap the element with the smaller child, and recurse on the child you swapped the element with. The recursion should stop when either you reach a leaf or when the element of the node is smaller than both its children.

```
bubbledown (Internal 10 (Internal 3 (Internal 6 (Leaf 7) (Leaf 11)) (Internal 4 (Leaf 8) (Leaf 6)))
(Leaf 12)) => Internal 3 (Internal 4 (Internal 6 (Leaf 7) (Leaf 11)) (Internal 6 (Leaf 8) (Leaf 1
0))) (Leaf 12)
bubbledown (Internal 5 (Internal 3 (Internal 6 (Leaf 7) (Leaf 11)) (Internal 4 (Leaf 8) (Leaf 6)))
(Leaf 12)) => Internal 3 (Internal 4 (Internal 6 (Leaf 7) (Leaf 11)) (Internal 5 (Leaf 8) (Leaf 6)))
(Leaf 12)
```

6. Using the Maybe monad of Haskell, create a function called checkcons that has the following type:

```
checkcons :: Maybe a -> Maybe [a] -> (a -> Bool) -> Maybe [a]
```

The function takes a Maybe value of some type, a Maybe list of the same type (as a monad), and a test function and returns a Maybe list of the same type. If either Maybe is Nothing, the result is Nothing. If the first Maybe value passes the test function, the result has the first element cons'd onto the front of the list. Otherwise the result is Nothing.

```
checkcons (Just 5) (Just [6,7,8]) (\x -> (x > 0))  =>  Just [5,6,7,8]
checkcons (Just 5) (Just [6,7,8]) (\x -> (x < 0))  =>  Nothing
```

7. Using checkcons create a function checklist that takes a list and a function and returns Nothing if the elements in the list fail to past the function and the list (embedded in a Maybe) if all the elements pass.

```
checklist "aaaaa" (\x -> x == 'a')
Just "aaaaa"

checklist "abcde" (\x -> (x >= 'a' && x <= 'z'))
Just "abcde"

checklist "abcDe" (\x -> (x >= 'a' && x <= 'z'))
Nothing

checklist [1,-2,3] (\x -> x > 0)
Nothing
```

8. In Haskell, lists are monads. As we saw in class, [1,2,3] >>= (\v -> [2*v]) produces the list [2,3,4]. In this problem, you are to figure out how that works.

Create a list monad that generalizes a list. This will not be a Haskell Monad type, but instead one of our own creation like the Value type from lecture. For example, the following is a valid "list".

```
Pair 4 (Pair 5 (Pair 6 Null))
```

Then create a binding function lbind and a return function lreturn to make a list monad. The code should work so that

```
(Pair 4 (Pair 5 (Pair 6 Null))) `lbind` (\x -> lreturn (2 * x))   =>   Pair 8 (Pair 10 (Pair 12 Nul
l))
```