Lund University
Department of Computer Science

EDAF95 (aka EDAF40)
1st June 2023, 8–13

Basics of Functional Programming Exam

1. **Point-free notation** (1p)

   Rewrite the following two definitions into a point-free form (i.e., `f = ...`, `g = ...`), using neither lambda-expressions nor list comprehensions nor enumeration nor `where` clause nor `let` clause:

   ```
   f x y = 3 - x / y
   g x y = [y z | z <- [1..x]]
   ```

2. **Type derivation** (1p)

   (a) (0.2p) Find the type of `map iterate`

   (b) (0.2p) Find the type of `curry uncurry`

   (c) (0.2p) Find the type of `uncurry curry`

   (d) (0.2p) Find the type of `curry . uncurry`

   (e) (0.2p) Find the type of `uncurry . curry`

3. **Functions** (1p)

   Redefine `map f` and `filter f` using `foldr`.

4. **Bind** (1p)

   Consider the following function:

   ```
   eliminate1 n [] g = Just g
   eliminate1 n (s:ss) g = eliminate n s g >>= eliminate1 n ss
   ```

   (a) (0.4p) Given that the type of `g` is `Grid`,

   ```
   g :: Grid
   ```

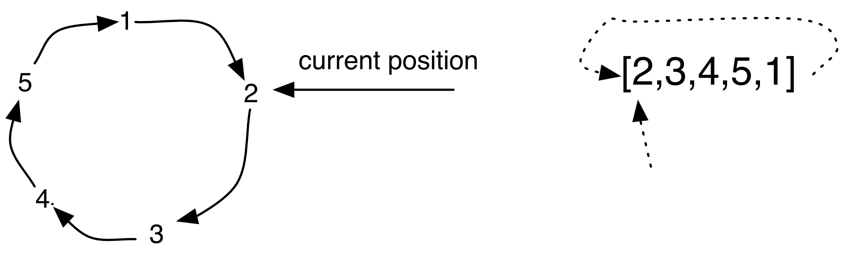
   write the signature for `eliminate1` . Assume the most generic type for `eliminate`.

   (b) (0.2p) How would your answer look like if the first line were changed to

   ```
   eliminate1 n [] g = [g]
   ```

   (c) (0.4p) Would the second line of the definition be correct after this change? Answer YES or NO, and motivate.

5. (2p) Define a type `CircList` (or `CL` for short, if you prefer) defining a circular list of arbitrary length (and holding arbitrary elements). Our examples below will use elements of type `Int`, but your solution should accommodate any type, even functions. You have to make sure that the *current position* is well-defined and accessible for the operations defined for this type. The picture below illustrates the concept and its possible representation using a standard list (with the assumption that the first element defines the current position and that the last position in the list is virtually glued to the first one in a circular fashion). Please note that you don't need to stick to this particular representation, it is just an illustration of the idea.



Please note that it is your task to define an appropriate type constructor! Define then for this type the following functions:

```
perimeter     :: CircList a -> Int
currentelem   :: CircList a -> Maybe a
nextelem      :: CircList a -> Maybe a
previouselem  :: CircList a -> Maybe a
insert        :: a -> CircList a ->  CircList a
delete        :: Int -> CircList a -> CircList a
takefromCL    :: Int -> CircList a -> [a]
mvCurrentelem :: CircList a -> CircList a
```

(1) returning the number of elements (positions) in the list; (2) returning the current element in the list; (3) returning the next element in the list; (4) returning the previous element in the list; (5) inserting an element between the current and the previous element in the list and setting the current element to the freshly inserted one; (6) deleting n first elements from the list; (7) taking n first elements of the circular list (possibly circling if necessary); and (8) shifting the current position forward by one element, respectively. You may, and are actually encouraged to, define any helper functions you deem appropriate. Examples of the intended functionality:

```
perimeter (CircList [1, 2, 3, 4, 5]) = 5
currentelem (CircList [1, 2, 3, 4, 5]) = Just 1
currentelem (CircList []) = Nothing
nextelem (CircList [1, 2, 3, 4, 5]) = Just 2
nextelem (CircList [1]) = Just 1
previouselem (CircList [1, 2, 3, 4, 5]) = Just 5
insert 6 (CircList [1, 2, 3, 4, 5]) = CircList [6, 1, 2, 3, 4, 5]
delete 2 (CircList [1, 2, 3, 4, 5]) = CircList [3, 4, 5]
takefromCL 4 (CircList [1, 2, 3]) = [1, 2, 3, 1]
mvCurrentelem (CircList [1, 2, 3, 4] = CircList [2, 3, 4, 1]
```

Finally, define a predicate (i.e., a function with boolean values)

```
equalCL :: CircList a -> CircList a -> Bool
```

yielding `True` if and only if both lists contain the same elements in the same order, but not necessarily with the same current position. E.g., if we used the above list representation for circular lists (assuming the end is glued to the beginning) then

```
equalCL (CircList [1, 2, 3, 4, 5]) (CircList [3, 4, 5, 1, 2]) = True
equalCL (CircList [1, 2, 3, 4, 5]) (CircList [3, 4, 5, 2, 1]) = False
equalCL (CircList [1, 2, 3, 4, 5]) (CircList [3, 4, 5, 1, 2, 3]) = False
equalCL (CircList [1, 2, 3]) (CircList [2, 3, 1, 2, 3, 1]) = False
```

Please explain why the last example needs to give the result `False`.

Good Luck!