Advanced Programming (I00032) 2016
Overloading and simple generics

Assignment 1

Preliminaries

There is an assignment for each lecture week. The assignments are mandatory and you may only skip one of them. The (strict) deadline is always just before the next lecture. Please, work with a partner, hand in one solution together on Blackboard and put your names and student numbers in the file. You do not get a mark, but assignments are checked and feedback is provided.

Goals of this exercise

This exercises review the implementation of instances of classes for various types. This means to implement functions, which have a similar meaning for the different types, and share the same name. In addition you will make a type constructor class and have to think about *kinds*.

Preparation

In this course we will use the functional programming language Clean. Clean is available for Windows, Linux and Max OS X and can be downloaded at `http://wiki.clean.cs.ru.nl/Download_Clean`. Please, do **not** use the version with *iTasks*. The IDE is unfortunately available for Windows only, but it is not really necessary for the assignments of this course (although it is rather useful).

On Linux and Mac you can use the comment line tool `clm`. For the first assignments you only need a single source file. The file `program1.icl` is compiled and executed by

```
clm program1 -o program1
./program1
```

For program2.icl uses the module `StdMaybe`, which is not part of the `StdEnv`, you have to add this module explicitly. To include it in the IDE select the environment `Everything`. You can also use `StdEnv` and manually add the module `StdMaybe` from the `StsLib`-directory. You have to add the path `{Application}\Libraries\StdLib` to your project. On the console on Linux or Mac you can build with:

```
clm -IL ../lib/StdLib program2
```

The Clean system comes with a language description in pdf. If you are new to functional programming in Clean you might like `http://www.mbsd.cs.ru.nl/papers/`

cleanbook/CleanBookI.pdf. The standard library StdEnv is described in http://www.mbsd.cs.ru.nl/publications/papers/2010/CleanStdEnvAPI.pdf.

We assume that you are familiar with the basics of functional programming in general, and Clean in particular. It is possible to repair a shortage of knowledge during this course, but this requires a considerable effort.

1 Type classes

A class defines a set of functions. There is special syntax for singletons (classes with a single function). Our example class serialize has two functions. The functions write translates a value to a list of strings. These strings can be easily written to a file or send over a serial communication channel. The function read does the inverse translation; it turn a list of Strings to a value. Since this conversion may fail, the result is of type Maybe (a,[String]) instead of (a,[String]).

```
class serialize a where
    write :: a [String] → [String]
    read  :: [String] → Maybe (a,[String])
```

The instance of this class for Booleans can be defined as:

```
instance serialize Bool where
    write b c = [toString b:c]
    read ["True":r] = Just (True,r)
    read ["False":r] = Just (False,r)
    read _ = Nothing
```

We can make a general function test that checks that reading de written list of strings yields the given value.

```
test :: a → (Bool,[String]) | serialize, == a
test a = (isJust r && fst jr==a && isEmpty (tl (snd jr)), s)
where
    s = write a ["λn"]
    r = read s
    jr = fromJust r

Start =
    [test True
    ,test False
    ]
```

Defines instances of the class serialize for the types Int, the predefines list [a], binary trees Bin, and Rose Trees Rose. The binary tree is defines as:

```
:: Bin  a = Leaf | Bin (Bin a) a (Bin a)
:: Rose a = Rose a [Rose a]
```

2 Kinds

Given the following type definitions:

```
::  Bin  a   = Leaf | Bin (Bin a) a (Bin a)
::  Tree a b = Tip a | Node (Tree a b) b (Tree a b)
:: Rose a   = Rose a [Rose a]
:: T1 a b   = C11 (a b) | C12 b
:: T2 a b c = C2 (a (T1 b c))
:: T3 a b c = C3 (a b c)
:: T4 a b c = C4 (a (b c))
```

What is the *kind* of the following types: `Bool`, `Bin`, `Rose`, `Bin Int`, `Tree`, `T1`, `T2`, `T3`, and `T4`? List the kinds as a comment in your code handed in on Blackboard.

3 Type Constructor Classes

The class `Container` is defined as:

```
class Container t where
    Cinsert   :: a (t a) → t a      | <        a
    Ccontains :: a (t a) → Bool     | <, Eq    a
    Cshow     ::   (t a) → [String] | toString a
    Cnew      :: t a
```

A container has elements of type `a`. There can be several implementations of `Container` with one uniform interface. A simple implementation of the container is just an unsorted list. A more advanced implementation is a binary search tree.

1. Give implementations of this container type constructor class for the list type [] and the binary tree type `Bin` from the first section.
2. Test the correctness of you implementations by evaluating expressions like:

   ```
   Start = (Ccontains 3 c, Cshow c) where c = ..
   ```

Deadline

The deadline for this exercise is September 6 2016, 13:30h (just before the next lecture).