HW 6 **Prof. Caldwell**
Due: 19 September 2006 **COSC 3015**

1

Problem 1.1. Read Chapter 2 of Bird.

Here is some Haskell code for a type `Boolean` that does not interfere with the builtin type `Bool`. I renamed `not` to `neg` so it does not conflict with the `not` in the Haskell type `Bool`.

```
module Boolean where

data Boolean = FF | TT             deriving (Eq,Ord,Enum,Show)

neg :: Boolean -> Boolean
neg FF  = TT
not TT = FF

(/\), (\/) :: Boolean -> Boolean -> Boolean

FF /\ x = FF
TT /\ x = x

FF \/ x = x
TT \/ x = TT
```

Problem 1.2. Do problems 2.1.1, 2.1.3, 2.1.6 on pages 34 – 35. For 2.1.3 use the operator `(.=>)` instead of `(=>)` or just use the prefix `implies`.

2 Type Classes

The deriving clause in the declaration of the type `Boolean` generates many usefull functions for the type. Here are the declarations of these classes.

2.1 `Eq` Typeclass

Described on page 31 of Bird.

```
class  Eq a  where
        (==), (/=)  ::  a -> a -> Bool

        x /= y  = not (x == y)
        x == y  = not (x /= y)
```

2.2 `Ord` Typeclass

Described on pg 32 of Bird.

```
class  (Eq a) => Ord a  where
    compare              :: a -> a -> Ordering
    (<), (<=), (>=), (>) :: a -> a -> Bool
    max, min             :: a -> a -> a

    compare x y | x == y    = EQ
                | x <= y    = LT
                | otherwise = GT

    x <= y  = compare x y /= GT
    x <  y  = compare x y == LT
    x >= y  = compare x y /= LT
    x >  y  = compare x y == GT

    -- Note that (min x y, max x y) = (x,y) or (y,x)
    max x y | x <= y    =  y
            | otherwise =  x
    min x y | x <= y    =  x
            | otherwise =  y
```

2.3 `Enum` Typeclass

Described on page 38 – 40 of Bird.

```
class  Enum a  where
    succ, pred     :: a -> a
    toEnum         :: Int -> a
    fromEnum       :: a -> Int
    enumFrom       :: a -> [a]            -- [n..]
    enumFromThen   :: a -> a -> [a]       -- [n,n'..]
    enumFromTo     :: a -> a -> [a]       -- [n..m]
    enumFromThenTo :: a -> a -> a -> [a]  -- [n,n'..m]

    -- Default declarations given in Prelude
```

2.4 `Show` Typeclass

Described on page 52 – 54 of Bird.

```
class  Show a  where
    showsPrec :: Int -> a -> ShowS
    show      :: a -> String
    showList  :: [a] -> ShowS

    showsPrec __ x s   = show x ++ s
    show x             = showsPrec 0 x ""
    -- ... default decl for showList given in Prelude
```

To find out more, read section 6.3 of of `http://www.haskell.org/onlinereport/basic.html`.

Problem 2.1. Write expressions that exercise the functions `( ==, /=, <, <=, >=, >, succ, pred, fromEnum, toEnum, show, showList)` generated automatically for the type `Boolean` by the Haskell deriving clause `(Eq,Ord,Enum,Show)`.