1

Problem 1.1. Read Chapter 1 and 2 of the Bird text.

2 info, infix operators and precedence

GHC will tell you the type, where the function is defined and, for infix operators, the precedence of the operator. Here are some examples.

```
*Main> :info (&&)
(&&) :: Bool -> Bool -> Bool    -- Defined in GHC.Base
infixr 3 &&
*Main> :info (||)
(||) :: Bool -> Bool -> Bool    -- Defined in GHC.Base
infixr 2 ||
*Main> :info (.)
(.) :: (b -> c) -> (a -> b) -> a -> c   -- Defined in GHC.Base
infixr 9 .
*Main> :l infinity
:l infinity
[1 of 1] Compiling Main             ( infinity.hs, interpreted )
Ok, modules loaded: Main.
*Main> :info infinity
:info infinity
infinity :: t   -- Defined at infinity.hs:1:0-7
*Main>
```

Precedence level is an integer from 0 to 9 with (9 being the highest precedence.) Function application associates to the left and has precedence 10 so, for example `f x y + 7` is `((f x) y) + 7`.

Haskell allows users to declare your own infix operators and whether they associate to the left (`infixl`) or right (`infixr`) or do not associate (`infix`)). You must also specify the precedence of the new operator.

For example, in class I presented the definitions of conjunction (`&&`) and disjunction (`||`) as follows:

```
False &&  x = False
True &&  x = x

False || x = x
True || x = True
```

This code will load, but when you try to use it you will need to specify which definition of `&&` you mean to use.

For example, after loading the code above I get the following behavior.

```
*Main> True && False
True && False

<interactive>:1:5:
    Ambiguous occurrence ‘&&’
    It could refer to either ‘Main.&&’, defined at hw4c.hs:7:6
                          or ‘Prelude.&&’, imported from Prelude
```

The error message is indicating that the name `&&` is ambiguous. You could fix this by specifying which of the two definitions of `&&` you intend.

```
*Main> True Main.&& False
False
```

To avoid conflict with the built-in functions we might have written the following code which declares two new infix operators that associated to the right.

```
infixr 3 &&&
(&&&) :: Bool -> Bool -> Bool
False &&& y = False
True &&& y = y

infixr 2 |||
(|||) :: Bool -> Bool -> Bool
False ||| y = y
True ||| y = True
```

Now we can compute as we expect.

```
*Main> True &&& False
False
```

Problem 2.1. Rewrite[1] the definitions of `&&&` and `|||` so that they behave just like `&&` and `||` but only using `if-then-else` and the Boolean constants. Your definitions should look something like the following:

```
x &&& y = ???
x ||| y = ???
```

The right side of your definition (`???`) must only use the constructs `True`, `False`, `if-then-else`, and the variables `x` and `y`.

Problem 2.2. Declare two new infix operators implementing logical implication (`==>`) and exclusive-or (`<+>`). (If you do not know the truth table for implication and exclusive-or, look at the expected output file.) Both operators should associate to the right. Implication should have precedence 0 (so that it binds weaker than the other connectives) and exclusive-or should have the same precedence as regular disjunction (`||`).

Check the linked expected output file to see the tests you should run.

[1]This is essentially problem 2.2.1 from Bird on page 34.