2 Basic types

1. Define (||), using pattern-matching; and similarly, (&&).

The definition of (||) from the standard prelude is

$$\begin{aligned} &\mathit{False} \mathbin{||} x = x \\ &\mathit{True} \mathbin{||} x = \mathit{True} \end{aligned}$$

Note that this is non-strict in the second argument, so that (for example)

$$(x == 0) \mathbin{||} (y \mathbin{/} x > 1)$$

is well-defined even when $x = 0$.

2. Define (&&) and (||) once *conditional expressions* (that is, **if**...**then**... **else**...). Note that there is more than one plausible way to do it, but only one correct way because of strictness.

Using conditional expressions, we could define

$$\begin{aligned} x \mathbin{\&\&} y &= \mathbf{if}\ x\ \mathbf{then}\ y\ \mathbf{else}\ \mathit{False} \\ x \mathbin{||} y &= \mathbf{if}\ x\ \mathbf{then}\ \mathit{True}\ \mathbf{else}\ y \end{aligned}$$

Notice that these are different from

$$\begin{aligned} x \mathbin{\&\&} y &= \mathbf{if}\ y\ \mathbf{then}\ x\ \mathbf{else}\ \mathit{False} \\ x \mathbin{||} y &= \mathbf{if}\ y\ \mathbf{then}\ \mathit{True}\ \mathbf{else}\ x \end{aligned}$$

because of strictness—that is, (&&) and (||) are not commutative. (Can you give contexts in which these definitions give different results?)

Now do the same using *guarded equations*.

Using guarded equations, we could write equivalently

```
x && y
    | x         = y
    | otherwise = False
  x || y
    | x         = True
    | otherwise = y
```

which again are different from the similar programs in which the guard depends on *y* instead of *x*.

3. Define a function *charToNum* that converts a digit character to its numeric equivalent; for example,

```
charToNum '3' = 3
```

(Hint: You will need the predefined function *ord* :: *Char* → *Int*. To use it, add "**import** *Data.Char*" at the top of your Haskell file.)

A pattern-matching definition of *charToNum* is simple but tedious:

```
charToNum :: Char → Integer
charToNum '0' = 0
charToNum '1' = 1
charToNum '2' = 2
charToNum '3' = 3
charToNum '4' = 4
charToNum '5' = 5
charToNum '6' = 6
charToNum '7' = 7
charToNum '8' = 8
charToNum '9' = 9
```

A better definition uses the function *ord*, as suggested:

```
charToNum c
  | '0' ⩽ c && c ⩽ '9' = ord c − ord '0'
```

4. Define a function *showDate* that takes three integers representing the day, month and year, and returns them formatted as a string (Hint: The (++) operator to appends two strings, and the *show* function converts a number to a string). For example:

   ```
   showDate 2 8 2004 = "2 August 2004"
   ```

   The simple definition of *showDate* uses these three components:

   ```
   showDay :: Integer → String
   showDay d = show d

   showYear :: Integer → String
   showYear y = show y

   showMonth :: Integer → String
   showMonth 1  = "January"
   showMonth 2  = "February"
   showMonth 3  = "March"
   showMonth 4  = "April"
   showMonth 5  = "May"
   showMonth 6  = "June"
   showMonth 7  = "July"
   showMonth 8  = "August"
   showMonth 9  = "September"
   showMonth 10 = "October"
   showMonth 11 = "November"
   showMonth 12 = "December"

   showDate :: Integer → Integer → Integer → String
   showDate d m y = showDay d ++ " " ++ showMonth m
                    ++ " " ++ showYear y
   ```

   (There are more elegant ways of defining *showMonth*.)

If that was too easy, make the day number an ordinal:

```
showDate 2 8 2004 = "2nd August 2004"
```

```
showDayOrdinal :: Integer → String
showDayOrdinal d = showDay d ++ suffix (d `div` 10) (d `mod` 10)
  where
    suffix 1 _ = "th"
    suffix _ 1 = "st"
    suffix _ 2 = "nd"
    suffix _ 3 = "rd"
    suffix _ _ = "th"
```