The aim of this practical is to introduce datatypes and typeclasses. Try to write the types of all your functions, so as to get used to classes and datatypes. It is also best to write your programs as literate scripts, with ample comments in between functions. Remember that programming requires practice – just by working through these practical sheets as we go along, you will find that you need not do much more work for the final exam.

1. Let’s start with datatypes . . .

   (a) Define a new datatype `ChessPiece` to enable to describe a chesspiece. For those not so familiar with chess, there are six different pieces in chess: king, queen, bishop, knight, rook and pawn. Similarly, define another datatype `DraughtsPiece` to define a draughts piece, which can be either (i) a normal piece or; (ii) a promoted piece.

   (b) When analysing a game, one is usually interested in (amongst other things), how much material one has on the board. Since some pieces are stronger than others, each piece is associated with a value. In chess, the values are: 9 for a queen, 5 for a rook, 3.5 for a bishop, 3 for a knight and 1 for a pawn (since losing the king means losing the game, one does not give a score to the king in this fashion — for the sake of this assignment, give the king a value of zero). In draughts, the values are: 1 for a normal piece, and 3 for a promoted piece. Define functions `valueChess` and `valueDraughts`, which given a piece return its value.

   (c) Any square on a general game board may be either (i) empty; or (ii) occupied by a black piece; or (iii) occupied by a white piece. Define a parametrised datatype `Square a`, which describes the state of a square on a board (where `a` is the type of pieces that can occupy that square). Note that there are various ways of encoding this. Choose whichever one you prefer.

   (d) Define the following functions (all of which take a value of type `Square a` as input): (i) `occupied` will tell you whether or not the given square is occupied; (ii) `piece` what piece lies on the square (use the `Maybe` type to define this, since the square may be unoccupied) and (iii) `owner` returns who is the owner of the piece on that square (similarly use the `Maybe` type). Note that these functions should work for both chess and draughts squares (in other words, define three, not six functions).

   (e) When analysing the material value of a game position, one counts gives positive values to the white pieces, and negative values to the black pieces (eg a black queen in chess would have a value of −9). An empty square is worth zero points. Define two functions `valueSquareChess` and `valueSquareDraughts`, which given a `Square ChessPiece` and `Square DraughtsPiece` (respectively), return the value of the given square.

   (f) A whole board position is simply eight by eight squares. Using lists and `Square a`, define a datatype `Board a` (not a type synonym) to describe a board, where `a` is the type of the pieces that can be placed on the board.

   (g) The material value of a full game board position, is simply the sum of the values of the individual squares. Define functions `scoreChess` and `scoreDraughts` to analyse a board positions.
2. Now, for some typeclasses . . .

(a) Make all the types you defined above instances of the Show typeclass.

(b) Define a new typeclass Valued, such that if type a is an instance of this class, it will have a function value of type a -> Double.

(c) Make ChessPiece and DraughtsPiece instances of this class.

(d) Whenever a is in class Valued, so is Square a (since we can give the value of a square — 0 for an empty square, and the value of the piece, positive or negative depending on its colour, otherwise). Define this in Haskell.

(e) Using the Valued class, define a new function score, which works like scoreChess or scoreDraughts depending on its parameter.

3. Finally, let’s make things a bit more interesting adding functionality to be able to describe the way pieces move . . .

(a) Define a datatype Direction, which contains all the principal 8 compass directions (N, NW, W, SW, S, SE, E, NE).

(b) The movement capability of a piece can be one of:
   i. One square in a particular direction, or
   ii. Any number of squares in a particular direction, or
   iii. Either of two given possible movement capabilities, or
   iv. One movement capability, followed by another.

Some examples: (i) a normal draughts piece can move one square in any diagonal direction — therefore its movement capability can be described as (one square northwest, or one square northeast, or one square southeast, or one square southwest); (ii) a knight moves moves two squares in a horizontal or vertical direction, then one square sideways — for simplicity let’s see its movement capability if it starts moving to the north: (one square north, followed by one square north followed by (either one square west or one square east)).

Define a datatype Movement to describe these capabilities, and use them to define the movement capabilities given in these examples.

(c) The position on a board can be given using its cartesian coordinates (as a pair of integers ranging from (1,1) to (8,8)). Define a function validMoves, which given a movement capability and the starting cartesian coordinates, returns a list of possible destination coordinates. For example, one can check where a knight can move to by asking:

   validMoves knightCapabilities (3,1)

   This should return the list [(1,2), (2,3), (4,3), (5,2)].