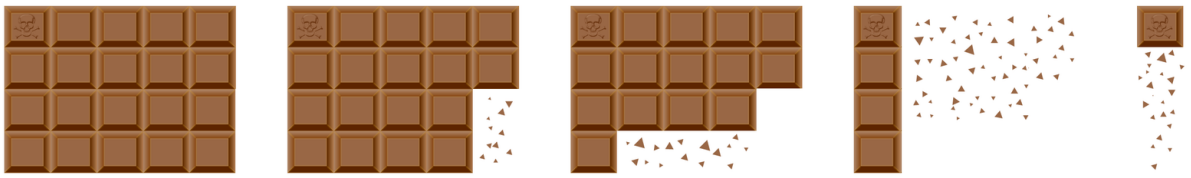
DM580 – EXERCISE SHEET #8

(1) Write a function `ioapply :: IO (a -> b) -> IO a -> IO b` that turns a IO action that returns a function into a function on IO actions.

(2) Write an IO action `adder :: IO ()` that accepts a natural number n followed by n integers separated by newlines from the prompt, and then writes back their sum, e.g.,

```
ghci> adder
How many numbers? 3
1
2
3
The total is 6.
```

(3) *Case: Chomp.* Chomp is a strategy game played by two players on a rectangular board of $n \times m$ cells, which are imagined to be the blocks of a chocolate bar. Players take turns choosing a block and "eating" it, removing that block *and all blocks below and to the right of it* from the board. The player who eats the top left block (imagined to be poisoned) loses the game. An example game sequence on a 4×5 board (via Wikimedia) is shown below.



The goal of this case is to implement this as an interactive game in Haskell, allowing two players to play Chomp by taking turns entering the coordinates of blocks to eat. You are free to do so however you wish – however, here are some suggestions.

(a) Come up types `Board` of Chomp boards, `Position` of positions on the board, and `Player` of players (e.g., Player 1 and Player 2).

(b) Make `Player` and `Board` instances of the `Show` type class such that `show board` (for some `board :: Board`) gives a reasonably nice textual representation of the given board state, e.g.,

```
* * * * *
* * * * *
* * * *
*
```

for the state of the board in the middle of the game above.

(c) Write a function `finished :: Board -> Bool` that returns whether the game has ended or not.

(d) Write a function `eat :: Board -> Position -> Board` that removes the blocks below and to the right of the given position from the given board.

(e) Write an IO action `getPosition :: IO Position` that queries the user for a position. *Hint:* Any type that is an instance of the `Read` type class can be read from a `String`.

(f) Write a function `turn :: Board -> Player -> IO (Board, Player)` that takes a state of the game (given by the current state of the board and the current player to play), queries the player for a position, plays it (if possible), displays the state of the game, and returns the new state of the game.

(g) Write an IO action `chomp :: IO ()` that puts it all together: displays the state of the game, allows players to take turns eating blocks (displaying the new state in between turns), and stopping once the game finishes, showing which player won.

(h) What happens if a player tries to play an invalid position? If you did not handle this case already, adjust your game accordingly.