

Chess

Informatics 1 – Functional Programming: Tutorial 1

Heijltjes, Wadler

Due: The tutorial of week 3 (9/10 Oct.)
Reading assignment: Chapters 4 and 5 (pp. 53–95)

Please attempt the entire worksheet in advance of the tutorial, and bring with you all work, including (if a computer is involved) printouts of code and test results. Tutorials cannot function properly unless you do the work in advance.

You may work with others, but you must understand the work; you can't phone a friend during the exam.

Assessment is formative, meaning that marks from coursework do not contribute to the final mark. But coursework is not optional. If you do not do the coursework you are unlikely to pass the exams.

Attendance at tutorials is obligatory; please let your tutor know if you cannot attend.

The ChessPieces module

In this tutorial we will be drawing pictures of chess pieces on a board. Download the package `tutorial1.zip` from the website and unzip it. The package contains three files:

`ChessPieces.hs` `Pictures.hs` `tutorial1.hs`

Open the last file, which contains this week's exercises, in emacs. Load it into GHCi and type this at the prompt:

```
Main> display knight
```

A window should appear displaying a picture of a white knight chess piece on a blue background:











The tutorial file `tutorial1.hs` gets the chess pieces from the file `ChessPieces.hs` by means of the first line of the file:

```
import ChessPieces
```

The file `ChessPieces.hs` in turn uses `Pictures.hs`, so you don't need to worry about that one. **Note:** If you get an error that GHCi can't find a module, you should put the files into the same directory (folder).

All in all the file `ChessPieces.hs` includes all chess pieces, white and grey squares to create a chessboard, and some functions to manipulate the images. The following tables show the basic pictures:

Chess pieces			Board squares		
bishop	A bishop		blackSquare	A black (grey) square*	
king	A king		whiteSquare	A white square	
knight	A knight				
pawn	A pawn				
queen	A queen				
rook	A rook				

* The black square is grey so that you can see the black pieces on it; but if you invert it, it becomes white (and the white square becomes grey).

And these are the functions for arranging them:

<code>flipV</code>	reflection in the vertical axis
<code>flipH</code>	reflection in the horizontal axis
<code>invert</code>	change black to white and vice versa
<code>superimpose</code>	place one picture onto another
<code>beside</code>	place one picture next to another
<code>above</code>	place one picture above another
<code>repeatH</code>	place several copies of a picture side by side
<code>repeatV</code>	stack several copies of a picture vertically

Amongst these are the functions `beside`, `above`, `invert`, `flipV` and `flipH` (you have read on these last two functions in chapter 1 of the textbook). Try applying these in various combinations to learn how they behave (for instance: what happens if you put pictures of different height side by side). Just as with the simple picture `knight`, you can see the modified pictures by using the `display` function (be patient with large pictures). You'll probably need some parentheses, for example:

```
Main> display (beside knight (flipV knight))
```

Exercises


1. Use the `knight` picture and the above transformation functions to create the following two pictures:



Feel free to use a convenient intermediate picture for the second image, but for the next exercise it is better if you don't do this for the first image.


The fourth function, `superimpose`, can place a piece on a square, like this:


```
Main> display (superimpose rook blackSquare)
```

The result should look like this: . You can use `superimpose` both ways around to put pieces on squares, but if you try to put pieces on top of other pieces, strange things may happen.

Exercises

2. We will create a function that takes a picture of a chess piece and puts four of them together, like the knights in the first picture of exercise (1). In other words, your function should work as in the following examples:

Main> display (fourPieces bishop) 

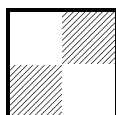
Main> display (fourPieces king) 

- (a) Write the function `fourPieces` that does this. **Note:** if you don't know how to start, take another look at the functions in the previous tutorial.
- (b) Try to predict what the following command will display:

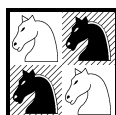
Main> display (fourPieces (fourPieces knight))

Then go ahead and see if you were right.


- 3. (a) Create a picture `fourSquares` of four chessboard squares like this:




- (b) Using the pictures and functions you have, and the function `superimpose`, create the following picture:



- (c) Now create a function `piecesOnSquares` that puts four pieces together on four squares, like this:

Main> display (piecesOnSquares rook) 

Main> display (piecesOnSquares queen) 

The functions `repeatH` and `repeatV` create a row or column of identical pictures, in the following way (try this out):

Main> display (repeatH 4 (superimpose (invert king) whiteSquare))

The full chessboard

Next, we will build a picture of a fully populated chessboard.

Notes:

- When a problem says “... using the function (or picture) `foo`,” you *must* use the function `foo`. A solution that does not use that function will not be accepted, but of course you can use other functions as well.
- Unless an exercise says you can't, you are free to define intermediate functions, or pictures in this case, if that makes it easier to define the solution to an exercise.

Exercises

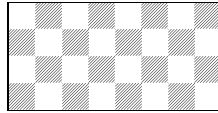
- 4. (a) Using the `repeatH` function, create a picture `emptyRow` representing one of the empty rows of a chessboard (this one starts with a white square).



- (b) Using the picture `emptyRow` from the last question, create a picture called `otherEmptyRow`, this time representing the *other* empty rows of a chessboard (this one starts with a grey square).



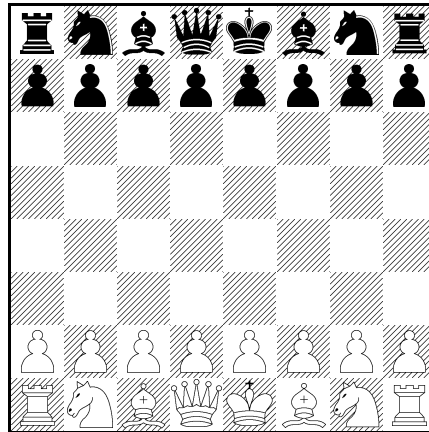
- (c) Using the previous two pictures, make a picture `middleBoard` representing the four empty rows in the middle of a chessboard:



- (d) Create a picture `whiteRow` representing the bottom row of (white) pieces on a chessboard, each on their proper squares. Also create a picture `blackRow` for the top row of (black) pieces. You can use intermediate pictures, but try to keep your knights pointing left. The pieces should look like this:



- (e) Using the pictures you defined in your answers to the questions above, create a fully-populated board (`populatedBoard`). It will be helpful to make pictures `blackPawns` and `whitePawns` for the two rows of pawns. The result should look like this:



Note: be patient while the program is drawing the board, it may take some time to appear in the window.

QuickCheck tests

At the end of the tutorial file `tutorial11.hs` you will find two QuickCheck functions:

```
prop_assoc_above
prop_invol_invert
```

Exercises

5. (a) Run the appropriate QuickCheck tests to verify these properties.
- (b) Look at the test functions and explain what they do, and why they are true.
- (c) Write a test function `prop_assoc_beside` that tests the same property (*associativity*) as `prop_assoc_above`, but then for the `beside` function.