Poker Hand

by ToF

November 10, 2011

$\lambda \quad \lambda \quad \lambda$

1 Solving the problem

Did we solve our problem?

Not yet.

What do we need to do, then?

Find the winner hand and mark that line in the game display.

What test should I write?

Write the simplest test you can think of.

Ok. Here it is:

```
showResults [TwoPairs] ~?= ["Two Pairs (winner)"]
```

We take a list of Rankings, and produce a list of displayed results. Here the list has only one line, so that line is the winner.

■ I see.

```
showResults :: [Ranking] → [String]
showResults [r] = [show r ++ " (winner)"]
```

■ That's easy.

Next, with two results:

```
showResults [Pair,TwoPairs] ~?= ["Pair", "Two Pairs (
    winner)"]
```

■ I'll do another *fake*.

```
showResults :: [Ranking] → [String]
showResults [r] = [show r ++ " (winner)"]
showResults [r,s] = [show r, show s ++ " (winner)"]
```

■ But we cannot continue this way.

What refactoring do you see?

I see a potential map

Go on.

I will move the *fake* part in the *mapped* function:

```
showResults :: [Ranking] → [String]
showResults = map showAndMark
    where showAndMark r | r == TwoPairs = show r ++ " (
        winner)"
          showAndMark r = show r
```

■ But we still have duplication.

Remove the duplication.

Ok:

```
showResults :: [Ranking] → [String]
showResults = map showAndMark
    where showAndMark r = show r ++ if r == TwoPairs
          then " (winner)" else ""
```

■ There.

Your function is called *showAndMark* so it does two things.

That's right.

Could you separate those two things in two functions?

I can try.

```
showResults :: [Ranking] → [String]
showResults = map showAndMark
    where showAndMark r = show r ++ mark r
          mark r = if r == TwoPairs then " (winner)"
              else ""
```

■ Not sure if it's what you asked for.

Can you use patterns instead of if then else?

Ok.

```
showResults :: [Ranking] → [String]
showResults = map showAndMark
    where showAndMark r = show r ++ mark r
          mark TwoPairs = " (winner)"
          mark _ = ""
```

■ There.

Good. Now get rid of *showAndMark*.

I can write it like this:

```
showResults :: [Ranking] → [String]
showResults = map (\r → show r ++ mark r)
    where mark TwoPairs = " (winner)"
          mark _ = ""
```

■ if it's what you mean.

Nice! Now here's another test.

```
showResults [Pair,HighCard] ~?=
    ["Pair (winner)","High Card"]
```

■ Ouch. Now we need to compute the real winner.

How do you find the winner?

■ That's the maximum from the list of Rankings.

```
showResults :: [Ranking] → [String]
showResults rs = map (\r → show r ++ mark r) rs
    where mark x | x == maximum rs = " (winner)"
          mark _ = ""
```

■ It works.

Could we avoid computing the maximum on each line?

You're right.

```
showResults :: [Ranking] → [String]
showResults rs = map (\r → show r ++ mark r) rs
    where mark x | x == max = " (winner)"
          mark _ = ""
          max = maximum rs
```

■ And I think we're done with showing results!

Let's not forget that we must display the input and the result. Here's a test:

```
showScore
   ["6♥ 6♦ 6♠ 6♣",
    "6♣ 4♦ A♣ 3♠ K♠",
    "6♣ 6♦ A♣ 3♠ K♠",
    "9♣ A♥ K♠ 3♣ K♦ 9♦ 6♦"] ~?=
      ["6♥ 6♦ 6♠ 6♣",
       "6♣ 4♦ A♣ 3♠ K♠ High Card",
       "6♣ 6♦ A♣ 3♠ K♠ Pair",
       "9♣ A♥ K♠ 3♣ K♦ 9♦ 6♦ Two Pairs (winner)"]
```

■ Wow, this is a big test.

And a decisive one, for that matter. Can you make it pass?

■ I think so. First, we have to compute the best Ranking for each line

```
showScore :: [String] → [String]
showScore input =
    let rankings = map maxRanking ss
```

Then we have to show the correspondant results:

```
        results = showResults rankings
```

Then we have to concatenate them with the input, using zipWith:

```
    in zipWith (++) input results
```

■ Ouch. It fails.

Sure it does:

```
Couldn't match expected type '[Ranking]'
against inferred type 'Maybe Ranking'
  Expected type: [[Ranking]]
  Inferred type: [Maybe Ranking]
In the second argument of 'map', namely 'rankings'
```

■ I see. We got it wrong with the type of results in the *showResult* function!.

What's wrong with it?

We can't describe the result for a line with no hand at all. So the function should be of type:
[Maybe Ranking] → [String] instead of
[Ranking] → [String].

That's right. Let's discard our test for now, and let me add this one:

```
showResults [Nothing,Just Pair] ~?= ["","Pair  (winner)"]
```

■ Of course, you have to adapt the remaining tests as well.