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?

Mark the lines from the input with the ranking of the hand, and suffix the bast one with "(winner").

What test should I write?

Write the simplest test you can think of.

What is the trivial case for a function that should mark lines?

No hand at all.

Ok

```
markResults [Nothing] ~?= [""]
```

If we don't have a hand, then there is no mark.

■ I see. Here's the function:

```
markResults :: [Maybe Ranking] → [String]
markResults _ = [""]
```

■ Done.

Here's my next case.

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

■ Ok. I'll just add a pattern:

```
markResults :: [Maybe Ranking] → [String]
markResults [Nothing]   = [""]
markResults [Nothing, Just Pair] = ["", "Pair  (winner)"]
```

■ It's a *fake*, as usual.

Do you see a possible refactoring here?

I see a map:

```
markResults :: [Maybe Ranking] → [String]
markResults = map mark
    where mark Nothing = ""
          mark (Just Pair) = "Pair  (winner)"
```

■ Refactoring done.

Here's a new case:

```
markResults [Nothing, Just Pair, Just HighCard] ~?=
                    ["","Pair (winner)","High Card"]
```

■ We can have several hands. The best one is the winner. There's *non-exaustive patterns* error in our code, now.

■ Sure. Here's a fix:

```
markResults :: [Maybe Ranking] → [String]
markResults = map mark
    where mark Nothing = ""
          mark (Just Pair) = "Pair (winner)"
          mark (Just r) = show r
```

■ It's still a *fake*.

How can we remove the *fake*?

By comparing each value in the list with the maximum value in the list.

```
markResults :: [Maybe Ranking] → [String]
markResults rs = map mark rs
    where mark Nothing = ""
          mark v@(Just r)
              | v == maximum rs = show r ++ " (winner)"
          mark (Just r) = show r
```

■ It works!

Can you remove duplication?

Yes.

```
markResults :: [Maybe Ranking] → [String]
markResults rs = map mark rs
    where mark Nothing = ""
          mark v@(Just r) = (show r) ++ if (v ==
                maximum rs) then " (winner)" else ""
```

■ Done.

Could we have pattern in lieu of the if then else?

Yes.

```
markResults :: [Maybe Ranking] → [String]
markResults rs = map mark rs
    where mark Nothing = ""
          mark (Just r) = (show r) ++ winner (Just r)
          winner v | v == maximum rs = " (winner)"
          winner _ = ""
```

■ Done.

And we could avoid computing the maximum at each line.

You are right.

```
markResults :: [Maybe Ranking] → [String]
markResults rs = map mark rs
    where mark Nothing = ""
          mark (Just r) = (show r) ++ winner (Just r)
          winner v | v == m = " (winner)"
          winner _ = ""
          m = maximum rs
```

■ And now we are done with marking results.

What else is missing?

Our program will have to reproduce and complete the input lines.

Ok. Here a test:

```
,scores ["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 an important one, for that matter. Can we make it pass?

Let's try. First We have to find the max ranking for each hand:

```
scores :: [String] → [String]
scores input = let  rs = map maxRanking input
```

Then we have to compute the marks:

```
ms = markResults rs
```

Then we join them with a concatenation operation:

```
in zipWith (++) input ms
```

■ Does it work?

No. The resulting lines lack a space between the input and the marks: We expect ... "6♣ 6♦ A♣3♠ K♠ Pair" ... and we get ... "6♣ 6♦ A♣3♠ K♠Pair"...

■ Then (++) is not the good operation to zip the lists with.

Do you know of a function which add spaces between strings?

Yes: unwords. For example, unwords ["THE","CAT","IN", "THE","HAT"] gives "THE CAT IN THE HAT".

Then let's use it.

Allright:

```
scores :: [String] → [String]
scores input = let  rs = map maxRanking input
                    ms = markResults rs
               in zipWith join  input ms
               where join a b = unwords [a,b]
```

■ Does it work now?

No. We have a supplementary space on the first line: We expect "6♥6♦ 6♠ 6♣" and get "6♥6♦ 6♠ 6♣ ".

■ In that case, let write a custom function.

```
scores :: [String] → [String]
scores input = let  rs = map maxRanking input
                    ms = markResults rs
               in zipWith join  input ms
               where join a "" = a
                     join  a b  = a ++ ' ': b
```

■ I should have thought of that in the first place!