# Poker Hand

by ToF

November 1, 2011

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

# 1 Printing

---

What should we work on, now ?

Let's do something that is easy, for a change.

---

What about printing the rankings ?

That will be short and sweet.

---

What do we print the ranking of ?

Hands.

---

Here's my first test

```
showRanking (hand "6♣ 4♦ A♣ 3♠ K♠") ~?= "HighCard"]
```

Here's the code to pass the test:

```
showRanking :: Hand → String
showRanking _ = "HighCard"
```

■ That was easy.

---

I immediately add another test:

```
showRanking (hand "6♣ 4♦ A♣ 3♠ 3♠") ~?= "Pair"]
```

I'll just add some patterns:

```
showRanking :: Hand → String
showRanking (HighCard _) = "HighCard"
showRanking (Pair _) = "Pair"
```

■ Easy. And wrong.

---

What's wrong with that code ?

We already have keywords for ranking values, so we just have to show these values.

---

You mean I should write my test this way:

```
      ,show HighCard ~?= "HighCard"
      ,show Pair ~?= "Pair" ]
```

Is that what you mean ?

■ Exactly.

---

But these tests provoke an error:

```
No instance for (Show ([Card] -> Hand))
  arising from a use of ‘show’ at Tests.hs
Possible fix: add an instance declaration
  for (Show ([Card] -> Hand))
```

It’s a problem because Pair and HighCard are of type "function from list of Cards to Hands". In fact we cannot add an instance declaration for this type.

---

What can we do to show those keywords then ?

Consider HighCard, Pair etc. as values of a type.

---

Ok, let’s do this.

First we create the new type:

```
data Ranking = HighCard
             | Pair
             | TwoPairs
             | ThreeOfAKind
             | Straight
             | Flush
             | FullHouse
             | FourOfAKind
             | StraightFlush
               deriving (Show)
```

■ Of course, these values conflict with the values declared in the Hand type.

---

That’s right. We have *multiple declarations* of all these values. We have to refactor.

Before refactoring we should first get back to the green.

---

You’re right. So I’ll delete my tests.

And I’ll remove my declaration of Ranking.
■ We’re back to green.

---

Now change the Hand type to include the Ranking.

Yes. I’ll replace the Hand type declaration:

```
data Hand = HighCard [Card]
          | Pair [Card]
          | TwoPairs [Card]
          | ThreeOfAKind [Card]
          | Straight [Card]
          | Flush [Card]
          | FullHouse [Card]
          | FourOfAKind [Card]
          | StraightFlush [Card]
            deriving (Ord,Eq)
```

With a new declaration. Now to declare a Hand we use a constructor, H, followed by a Ranking and a list of Cards.

```
data Hand = H Ranking [Card]
          deriving (Ord, Eq)

data Ranking = HighCard
             | Pair
             | TwoPairs
             | ThreeOfAKind
             | Straight
             | Flush
             | FullHouse
             | FourOfAKind
             | StraightFlush
               deriving (Eq, Ord, Show)
```

■ This shouldn’t work yet, though.

Exact, we have many errors:

```
Couldn't match expected type '[Card] -> Hand'
  against inferred type 'Ranking'
```

■ We have to change the *ranking* function:

```
ranking :: [[Card]] → Hand
ranking [[a,b,c,d],[e]] = H FourOfAKind [a,b,c,d,e]
ranking [[a,b,c],[d,e]] = H FullHouse [a,b,c,d,e]
ranking [[a,b,c],[d],[e]] = H ThreeOfAKind [a,b,c,d,e]
ranking [[a,b],[c,d],[e]] = H TwoPairs [a,b,c,d,e]
ranking [[a,b],[c],[d],[e]] = H Pair [a,b,c,d,e]
ranking [[a],[b],[c],[d],[e]] = H HighCard [a,b,c,d,e]
```

And those two functions as well:

```
promoteStraight :: Hand → Hand
promoteStraight (H HighCard [a,b,c,d,e])
    | value a − value e == 4 = H Straight [a,b,c,d,e]
promoteStraight (H HighCard [a,b,c,d,e])
    | value a == 14 && value b == 5 = H Straight [b,c,d,e,
        a]
promoteStraight h = h

promoteFlush :: Hand → Hand
promoteFlush (H HighCard cs) | flush cs = H Flush cs
promoteFlush (H Straight cs) | flush cs = H StraightFlush
    cs
promoteFlush h = h
```

■ And now everything is working.

---

But how do we print the Ranking value of a Hand ?

Write a test.

---

Ok, I'll just restore my first test:

```
show (rank (hand "6♣ 4♦ A♣ 3♠ K♠")) ~?= "HighCard"
```

■ I named the function *rank* because the name *ranking* is already used.

■ Ah, but we can change this.

---

First, make the test pass.

■ You are right.

```
rank :: Hand → Ranking
rank (H r _) = r
```

■ Done.

---

Now I change my test

```
show (ranking (hand "6♣ 4♦ A♣ 3♠ K♠")) ~?= "
    HighCard"
```

■ And you do the renaming.

■ Ok, first the function giving the Ranking of a hand:

```
ranking :: Hand → Ranking
ranking (H r _) = r
```

---

■ Then the function to compute the ranking from the card groups:

```
rank :: [[Card]] → Hand
rank [[a,b,c,d],[e]] = H FourOfAKind [a,b,c,d,e]
rank [[a,b,c],[d,e]] = H FullHouse [a,b,c,d,e]
rank [[a,b,c],[d],[e]] = H ThreeOfAKind [a,b,c,d,e]
rank [[a,b],[c,d],[e]] = H TwoPairs [a,b,c,d,e]
rank [[a,b],[c],[d],[e]] = H Pair [a,b,c,d,e]
rank [[a],[b],[c],[d],[e]] = H HighCard [a,b,c,d,e]
```

Then the main function:

```
hand :: String → Hand
hand = cards
        >>. rSortBy (comparing value)
        >>. groupBy (same value)
        >>. rSortBy (comparing length)
        >>. rank
        >>. promoteStraight
        >>. promoteFlush
```

■ The refactoring is done.