School of Computer Science, University of Nottingham

G51FUN Functional Programming, Spring 2015

Graham Hutton

Coursework II – The Winner Takes It All

Tuesday 10th March 2015

Deadline: Monday 27th April 2015, 11am

The next general election in the United Kingdom takes place on 7th May 2015. A few years ago there was nationwide referendum to decide whether to replace the simple *first past the post* system for general elections with the more refined *alternative vote* system. The decision was strongly in favour of retaining the current system. The aim of this coursework is to implement these two voting systems in Haskell, using the material from Lectures 1-7.

The coursework is worth 15% of the module mark, and may either be solved on your own, or jointly with ONE other student taking the module. Larger teams are not permitted. Half of the marks are for correctness, and half for programming style. In particular, you should aim to make your definitions as simple, clear and elegant as possible. You should complete the coursework in your own time, but if you get stuck you can ask for help during the lab sessions.

Assessment will be carried out by oral examination during the lab sessions (nothing needs to be handed in.) When you have completed the coursework you should ask a tutor to examine your solution. The tutor will then ask you some questions to test your understanding. You will not receive any marks if you cannot explain your solution. If your team has two members, both members must be present at the time of assessment. Absent team members will not receive any marks. If you are unable to successfully complete some of the definitions then comment out those that are incomplete. Note that tutors will only be available to assess your solution during the official lab session (Mondays, 9am to 11am, A32), and that you can only be assessed once.

Your script should begin with the following declarations:

```
import Data.List (sort)
type Party = String
type Ballot = [Party]
```

The first line imports the library function *sort* :: *Ord a* ⇒ [*a*] → [*a*] that sorts a list of values, provided that the values have an ordering. The remaining lines declare a party name as a string, and a ballot paper as a list of preferences (first choice, second choice, etc.)

First Past The Post

In this voting system, each person has one vote, and the party with the largest number of votes is the winner. We will build up to implementing this system in a number of steps.

1. Define a function

   ```
   count :: Eq a ⇒ a → [a] → Int
   ```

   that counts the number of occurrences of a value in a list, provided that the values support equality. For example, *count* `'a'` `"ababca"` should return 3.

2. Define a function

   ```
   rmdups :: Eq a ⇒ [a] → [a]
   ```

   that removes duplicates from a list. For example, *rmdups* `"ababca"` should return `"abc"`.

3. Using *count* and *rmdups*, define a function

   ```
   frequency :: Eq a ⇒ [a] → [(Int, a)]
   ```

   that counts how many times each distinct value in a list occurs in that list. For example, *frequency* `"ababca"` should return [(3, `'a'`), (2, `'b'`), (1, `'c'`)].

4. Using *sort* and *frequency*, define a function

   ```
   results :: [Party] → [(Int, Party)]
   ```

   that takes a list of all the votes that were cast (one per person) and returns the results of the election in increasing order of the number of votes. For example, if we define

   ```
   votes :: [Party]
   votes = ["Red", "Blue", "Green", "Blue", "Blue", "Red"]
   ```

   then *results votes* should return

   ```
   [(1, "Green"), (2, "Red"), (3, "Blue")]
   ```

   If more than one party has the same number of votes, they should be returned in alphabetical order. *Hint*: try some experiments to see what *sort* does on lists of pairs.

5. Using *results*, define a function

   ```
   winner :: [Party] → Party
   ```

   that takes a list of votes and returns the winner under the first past the post scheme. For example, *winner votes* should return `"Blue"`.

Alternative Vote

In this voting system, each person can vote for as many or as few parties as they like, listing them in preference order on their ballot paper (first choice, second choice, etc.) To decide which party wins, we begin by eliminating the party with the smallest number of first preference votes, and then repeat this process until only one party remains. This party is the winner.

6. Define a function

   $$\mathit{rmempty} :: \mathit{Eq}\ a \Rightarrow [[a]] \to [[a]]$$

   that removes all occurrences of the empty list from within a list of lists. For example, *rmempty* `["abc", "", "bc", ""]` should return `["abc", "bc"]`.

7. Define a function

   $$\mathit{remove} :: \mathit{Eq}\ a \Rightarrow a \to [[a]] \to [[a]]$$

   that removes all occurrences of a given value from within a list of lists. For example, *remove* `'a'` `["abc", "bc", "aa"]` should return `["bc", "bc", ""]`.

8. Using *results* from the previous page, define a function

   $$\mathit{rank} :: [\mathit{Ballot}] \to [\mathit{Party}]$$

   that takes a list of all the ballot papers that were submitted (one per person), selects the first preference vote on each ballot, and returns the result of a first past the post election based upon these votes, in increasing order of the number of votes. You may assume that each ballot is non-empty, i.e. contains at least one vote. For example, if we define:

   ```
   ballots :: [Ballot]
   ballots = [b1, b2, b3, b4, b5, b6]

   b1 = ["Blue", "Green"]
   b2 = ["Green", "Blue", "Red"]
   b3 = ["Blue"]
   b4 = ["Red", "Green"]
   b5 = ["Blue", "Red", "Green"]
   b6 = ["Green", "Red"]
   ```

   then *rank ballots* should return `["Red", "Green", "Blue"]`, because in terms of first preferences, `"Red"` has one vote, `"Green"` has two, and `"Blue"` has three.

9. Fill in the missing parts ??? in the function

   ```
   election :: [Ballot] → Party
   election bs = case rank (rmempty bs) of
     [p]      → ???
     (p : ps) → ???
   ```

   that takes a list of ballot papers and returns the winner under the alternative vote scheme. For example, *election ballots* should return `"Green"`.