COMP 212 Fall 2024
Homework 06

1 Shopping Cart

Your goal is to implement a simple shopping cart, using the same ideas as last Thursday's lecture: we represent the states of an interactive application by the constructors of a `datatype`

An interaction with your shopping cart should look roughly like this:

```
Please enter your name:
[you type] Dan
Hi, Dan What would you like to buy?
  apples  $1/pound
  bananas $2/bunch
  cookies $2/box
Or say 'checkout' to check out.
[you type] apples
Hi, Dan What would you like to buy?
  apples  $1/pound
  bananas $2/bunch
  cookies $2/box
Or say 'checkout' to check out.
[you type] bananas
Hi, Dan What would you like to buy?
  apples  $1/pound
  bananas $2/bunch
  cookies $2/box
Or say 'checkout' to check out.
[you type] apples
Hi, Dan What would you like to buy?
  apples  $1/pound
  bananas $2/bunch
  cookies $2/box
Or say 'checkout' to check out.
[you type] checkout
Hi, Dan
Your cart contains apples,bananas,apples.
```

```
I will charge you $4.
Type 'pay' to pay.
[you type] pay
Please enter your name:
...
```

The key features are:

- At the start, the application should ask for the user's name, which should be displayed while they are shopping.
- The application should repeatedly ask the user to select something to buy, which they can choose by typing the name of the product.
- Each product should have a price.
- Typing `checkout` should take the user to a screen that lists the items they have selected along with the total price.
- Then typing `pay` should take the user back to the beginning (with the idea that it has paid for the order).

You can choose what products are for sale, how much they cost, and exactly how things are displayed. For simplicity, the list of products and prices can be fixed/hard-coded into the code, though in reality this would be pulled from a database that is updated as people buy things.

Task 1.1 (10 pts). Choose constructors for the datatype `model` to represent each state of the application, and pick arguments to the constructors that store the information associated with each state. For example, during the shopping phase, you will need to choose a representation for the shopping cart.

Task 1.2 (10 pts). Write a function `view : model -> string` that displays the model as a string.

Task 1.3 (10 pts). Write a function `respond : model * string -> model` that updates the model state based on user input.

See the instructions in `hw06.sml` for how to run your application using the `run()` function.

2 Non-collaborative problem

Remember that non-collaborative problems are to be done independently. You are not allowed to communicate with anyone about the problems, except to ask the instructor or TAs clarification questions (not hints). Additionally, you are not allowed to search for help on the specific problem from any sources besides the course materials.

Note: this problem can be solved using the techniques covered up through HW4 — you don't need to use trees or higher-order functions.

You are working on a new election prediction website. For this problem, we will represent the result of a political poll for a US Presidential election by a tuple

> (pollster name, year of election, Democratic candidate vote share, Republican candidate vote share)

For example, the tuple `("Napoleon",1980,39.18,51.18)` means that a polling firm named Napoleon polled the 1980 Presidential election between Jimmy Carter (Democrat) and Ronald Reagan (Republican) and found 39.18% of voters planned to vote for Carter, and 51.8% planned to vote for Reagan. The type of such tuples is

```
type poll = string * int * real * real
```

(The numbers may not add up to 100% because of other candidates, people who do not plan to vote, etc.)

A list of polls represents the results of multiple polls by multiple polling firms, for example

```
[("Evening Ask",1980,42.66,50.29),
 ("Division/Maroon",1980,40.92,50.43),
 ("Napoleon",1980,39.18,51.18)]
```

represents three (fake) polls for the 1980 election by three (fake) polling firms, named Evening Ask, Division/Maroon, and Napoleon.

Task 2.1 (5 pts). One way to use multiple polls to get a sense for an electoral race is to average them. Write a function `average` that takes a list of polls (all for the same election) and produces two numbers (d, r) where d is the average of the Democratic candidate's vote share, and r is the average of the Republican candidate's vote share.[1] For example, the output on the above three polls should be (40.92,50.6333333333).

Next, you will implement a simple way to adjust for "house effects". The idea of a house effect is that a particular pollster's methodology (e.g. who they contact, how they model whether someone will vote) might consistently overstate or understate support for a particular party's candidate. If you think a pollster is consistently missing the true vote share percentage in the same direction, you can correct for this.

For this problem, we use the following simplified approach:[2] Suppose you have some past polls by a polling firm, along with the actual true results of those past elections (we will represent an actual election result by a poll whose polling firm is named "Actual"). Then we can estimate how much a pollster is likely to miss the true results by because we can average their past misses. For example, if there were 3 past elections, and the pollster's Democratic candidate vote shares were $55\%, 50\%, 45\%$, but the true actual vote shares were $52\%, 55\%, 47\%$, then the pollster missed by $+3, -5, -2$, and the average of those is -1.33.

[1] The average of a list of numbers is the sum of the numbers divided by the number of numbers.

[2] This is not what people actually do, but the actual algorithm seemed too hard for a non-collaborative problem. Think about why this might or might not this be a good approach.

Thus, when we see a new poll from that pollster with Democratic vote share 45, we might adjust that to 46.33 because we know the pollster has historically underestimated the vote share for the Democratic candidate.

Task 2.2 (20 pts). Write a function `adjust` that takes (1) a list of polls for the current election, (2) a list of historical polls, and (3) a list of actual results for historical elections, and produces a list with each poll for the current election adjusted by the average miss for each pollster for each candidate, as determined by the historical polls and actual results. You will likely want to write some helper functions. You don't need to try to minimize the running time (work) of your function (we will talk about some data structures that can make this process more efficient after fall break). The variables `RESULTS`, `HISTORICAL`, and `CURRENT` provide some fake polling data for testing.

Task 2.3 (10 pts). In comments near the code, give recurrences and big-O for the work of `adjust` (and any helper functions you write).