1 Interaction Trees

This assignment explores several variations on the interaction trees example covered in the lecture.

The basic definition of interaction trees we start from is the following:

```
data ITree a
  = Ask String (Bool -> ITree a)
  | Result a
```

A basic interaction is:

```
burger_price :: ITree Float
burger_price =
  Ask "Do you want a big whopper?"
   (\ big -> if big
               then -- big whopper
                  Ask "Do you want fries with that?"
                   (\ fries -> if fries
                                 then Result 8.5
                                 else Result 5.5
                   )
               else -- small whopper
                  Ask "Do you want it in a happy kids box?"
                   (\ box -> if box
                                 then Result 10.0
                                 else Result 4.5
                   )
   )
```

This comes with the following basic interpreter:

```
ieval0 :: ITree a -> [Bool] -> a
ieval0 (Result r) answers  =  r
ieval0 (Ask q ts) (b:as)   =  ieval0 (ts b) as
```

An example interaction, ordering a big whopper with fries, results in the price:

```
> ieval0 burger_price [True,True]
8.5
```

Assignment 1: Write a variant of `ieval0` that does not crash when there are fewer answers (booleans) than questions:

```
ieval1 :: ITree a -> [Bool] -> Maybe a
```

Use the `Maybe` datatype (predefined in the automatically imported `Prelude` library) to distinguish between the cases where you have `Just` a rersult, and those where `Nothing` is returned for lack of sufficient answers.

```
data Maybe a = Nothing | Just a
```

The example interaction would behave as follows:

```
> ieval1 burger_price [True,True]
Just 8.5
> ieval1 burger_price [True]
Nothing
```

Assignment 2: Write a variant of `ieval1` that returns the list of the questions asked alongside the possible result.

```
ieval2 :: ITree a -> [Bool] -> ([String],Maybe a)
```

The example interaction would behave as follows:

```
> ieval2 burger_price [True,True]
(["Do you want a big whopper?","Do you want fries with that?"],Just 8.5)
> ieval2 burger_price [True]
(["Do you want a big whopper?","Do you want fries with that?"],Nothing)
```

Assignment 3: Write a mini-interaction that just returns the given result.

```
result :: a -> ITree a
```

This interaction would behave as follows:

```
> ieval2 (result 5) [True,True]
([], Just 5)
> ieval2 (result "Donald McRonald") []
([],Just "Donald McRonald")
```

Assignment 4: Write a mini-interaction that just returns ask the given question and just returns its answer.

```
ask :: String -> ITree Bool
```

This interaction would behave as follows:

```
> ieval2 (ask "Do you want mayonnaise?") [True]
(["Do you want mayonnaise?"], Just True)
> ieval2 (ask "Do you want mayonnaise?") [False]
(["Do you want mayonnaise?"], Just False)
> ieval2 (ask "Do you want mayonnaise?") []
(["Do you want mayonnaise?"], Nothing)
```

Assignment 5: Add support for `tell`ing the customer something without expecting an answer. This requires extending the `ITree` type with another constructor, adding a case to `ieval2` and writing a mini-interaction that just tells the customer the given message and returns the unit value.

```
tell :: String -> ITree ()
```

This interaction would behave as follows:

```
> ieval2 (tell "Thank you! Please come again!") []
(["Thank you! Please come again!"], Just ())
```

Assignment 6: The following function extends a given interaction tree. Add supporting the "telling" feature of the previous assignment.

```
extend :: ITree a -> (a -> ITree b) -> ITree b
extend (Result x) f = f x
extend (Ask q t)  f = Ask q (\ b -> extend (t b) f)
```

This should enable the following behavior:

```
> ieval2 (burger_price `extend` \ _ -> tell "Thank you!") [True,True]
(["Do you want a big whopper?","Do you want fries with that?","Thank you!"],Just ())
```

Assignment 7: Provide a version of `ieval2` that uses `IO` to have a dynamic interaction with the user.

```
ieval7 :: ITree a -> IO a
```

You may want to use functions like `readLn`, `getLine`, `putStr`, `putStrLn` and `print`.

```
> ieval7 (burger_price `extend` \ p -> tell ("Please pay " ++ show p ++ "€"))
Do you want a big whopper?
yes
Do you want fries with that?
hmmm
Do you want fries with that?
yes
Please pay 8.5€
```