COMP 212 Spring 2022
Final Non-Collaborative Challenge Problems

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

1 Programming and Analysis

Wordle[1] is a game that was popular a long time ago in early 2022.

The idea of the game is to guess a hidden 5-letter word by guessing 5-letter words. After each guess, the game tells the player which letters were correct, which were in the word but misplaced, and which were not in the word.

For example, if the word is "STAND", then for the guess "STAMP" the "S" and "T" and "A" are correct and the "M" and "P" are not in the word.

For this problem, you will implement a solver for a simplified version of Wordle, where for each letter in the guess the game replies with "Right" (meaing the letter in that position matches the hidden word) or "Wrong" (meaning the letter in that position does not match the hidden word).[2]

We will represent a word as a sequence of strings, with each string representing one letter:

```
(* each string is a single capital letter *)
type word = string Seq.seq
```

Letters will always be written in capitals. In the test data a word will always have 5 letters, but for full credit your code shouldn't dependent on this. The function

```
(* E.g. makeWord "ERROR" = <"E","R","R","O","R"> *)
makeWord : string -> word
```

converts a string to a sequence of letters for easy input.

A result is Right/Wrong

[1] https://www.nytimes.com/games/wordle/index.html

[2] For significant extra credit, you can modify all of the code to work for "misplaced" as well.

```
datatype result =
    Right
  | Wrong
```

A guess's result is a sequence of letters, each paired with whether they were right or wrong:

```
type guess_result = (string * result) Seq.seq
```

For example, the guess "STAMP" for hidden word "STAND" would result in the sequence

```
<("S",Right), ("T",Right), ("A",Right), ("M",Wrong), ("P",Wrong) >
```

The provided function

```
(* Checks a guess against a hidden word answer and returns the result.

   E.g. check(makeWord "STAND", makeWord "STAMP") =
        <("S",Right), ("T",Right), ("A",Right), ("M",Wrong), ("P",Wrong) >
*)
fun check(answer : word, guess : word) : guess_result
```

checks the letters in a guess against the letters in the answer.

The support code includes a sequence of around 2000 5-letter words in the variable `words`. You can assume that all hidden words and all guesses come from this list.[3]

1.1 Possibilities

Your solver should maintain a sequence of possible answers, which will originally be the 2000ish `words`. Each guess restricts the remaining possible answers by giving you information about, for each letter, whether the guess matches the answer.

Task 1.1 (10 pts). Write a function

```
fun update(possibilities : word Seq.seq, guess : guess_result) : word Seq.seq =
```

that takes a sequence of possible answers and a guess result and returns the sequence of remaining possible answers.

Task 1.2 (5 pts). Analyze the work and span of your `update` function. For the analysis, you can assume all words have 5 letters. If we haven't talked about the running time of a sequence helper function, you can assume that it is the same running time as for the implementation of that helper function for Node/Empty/Leaf trees.

[3]The actual Worlde uses only a subset of the guessable words as answers, but here we assume they are drawn from the same list.

1.2 Suggestions

Task 1.3 (15 pts). Write a function

```
(* Given a sequence of possible answers, pick one to guess *)
fun suggest(possibilities : word Seq.seq) : word
```

that picks a guess from a sequence of possible words. The algorithm for choosing the guess is up to you. Solutions that use fewer guesses to guess the hidden word (to win the game) will receive more credit.

1.3 Compiling

The support code includes a file `sources.cm` that attempts to import all of the sequence, dictionary, and extractcombine libraries from from previous assignments. (If you didn't finish a part and need a reference solution, just let me know.) This file assumes that your directory structure looks like this:

```
<comp212 folder>/src/
<comp212 folder>/final-handout/
<comp212 folder>/hw10-handout/
```

If it doesn't (e.g. you have `hw10-handout` inside your own `HW10` folder), you will need to fiddle with the paths to tell it where to look or copy the folders into your final folder to look like the above. If you need help with this let me know.

When you're testing, you may want to compile the code using

```
- CM.make "sources.cm"; open Wordle;
```

(The `open` will save you from having to prefix all of the functions with `Wordle.function` when you test.)

1.4 Testing

The function

```
fun play(answer : word, possibilities : word Seq.seq, shouldPrint : bool) : int
```

uses your `update` and `suggest` to repeatedly play the game until it wins, and returns the number of guesses made.

For example,

```
play(makeWord "STAND", words, true);
Guessing SLATE
Guessing STARK
Guessing STAID
Guessing STAND
You guessed it!
val it = 4 : int
```

means that the hidden word was "STAND", the word list `words` was the initial possibilities, and the program guessed "SLATE", "STARK", "STAID", "STAND" in that order and won in 4 guesses. (The boolean `true` controls whether the guesses get printed out as it goes or not.)

You can use `play` to see how your solver plays on some examples. To gather some bigger data about its performance, the function

```
fun histogram ()
```

plays the game on all 2000ish `words` and prints a histogram of the number of guesses needed to win each game. For example, in the chart

```
Num Guesses (N) | Games won in N guesses | % won in N guesses | % won in <=N guesses
.
.
.
.
6               | 559                    | 24.13              | 75.57
.
.
.
```

the row for 6 means that 559 games we were won in exactly 6 guesses, which is 24.13% of the 2317 words, and that the cumulative percentage won in 6 or fewer guesses is 75.57% of the 2317 words. (For calibration, my best solver gets around 25% in 4 or fewer guesses, 50% in 5 or fewer, 75% in 6 or fewer, and 90% in 7 or fewer).

Task 1.4 (5 pts). The main step in the histogram code is frequency-counting the numbers:

```
fun hist(s : int Seq.seq) : (int * int) Seq.seq =
    Dict.toSeq(ExtractCombine.extractcombine (Int.compare,
                                              fn c => Seq.singleton(c, 1),
                                              Int.+,
                                              s))
```

Analyze the work and span of `hist` for your dictionary and `extractcombine` implementation from Homework 10.

2 Proof

For this problem, we use trees with data at the leaves (as in Homework 7):

```
datatype 'a tree = Empty | Leaf of 'a | Node of 'a tree * 'a tree
```

Sometimes, a computation will produce a tree with patterns like `Node(Empty,t)` or `Node(t,Empty)` in it, which can be shrunk to just `t` without changing the contents of the tree. The following function does this:

```
fun shrink (t : 'a tree) : 'a tree =
    case t of
        Empty => Empty
      | Leaf x => Leaf x
      | Node (l,r) => (case (shrink l, shrink r) of
                            (Empty, r') => r'
                          | (l',Empty) => l'
                          | (l',r') => Node(l',r'))
```

That is, if either subtree of a tree shrinks to the empty tree, we delete that node, and otherwise we make a node of the shrunken subtrees.

In homework, you implemented `reduce` for trees.

```
fun reduce (n : 'a * 'a -> 'a, e : 'a, t : 'a tree) : 'a =
    case t of
        Empty => e
      | Leaf x => x
      | Node (l,r) => n (reduce (n, e, l), reduce (n, e, r))
```

In this problem, you will prove that the behavior of `reduce` is unchanged by shrinking:

Theorem 1. *Suppose we have total* `n:'a * 'a -> 'a` *and* `e:'a` *such that*

1. *For all* `x`, `n(e,x)` $=$ `x`
2. *For all* `x`, `n(x,e)` $=$ `x`

Then for all `t`, `reduce (n, e, shrink t)` $=$ `reduce (n, e, t)`

The assumptions say that the function n returns the other input when given e as one of its arguments (e.g. if n is $+$ and e is 0, or if n is $\times$ and e is 1).

Task 2.1 (15 pts). Prove this theorem.