Advanced Programming (CIS 552)
Homework Assignment 2

Due Monday, January 28, at 3PM

Warm-up

1. Read chapters 5 and 7 of SOE.
2. Work problem 5.2 from SOE (on paper — no need to turn this one in).
3. Work problems 5.3, 5,4, 5.5, 5.6, 7.1, and 7.2 from SOE (and turn them in as described below).
4. Show how to define `map` in terms of `foldr` (ditto).

Preliminaries

This assignment involves transforming XML documents. To keep things simple, we will not deal with the full generality of XML, or with issues of parsing. Instead, we will represent XML documents as instances of the following simplified type:

```
data SimpleXML =
         PCDATA  String
       | Element ElementName [SimpleXML]
     deriving Show

type ElementName = String
```

That is, a `SimpleXML` value is either a `PCDATA` ("parsed character data") node containing a string or else an `Element` node containing a tag and a list of sub-nodes.

A collection of files that form the starting point of this assignment is available on the main course web page. Begin by grabbing these files, unpacking them, and making sure that you can successfully run the main program (in `Main.hs`). We've provided a `Makefile`, which you can use if you like. You should see this output:

```
Converting... Results differ: 'WRITE ME!' vs '<html><body><h1>A Mi'
```

Next, have a look at the provided files.

- `Main.hs` is the main program — it's just a test harness for the rest.
- `Mine.hs` is your part. This is where you will put the code you write, and it's the only file you will turn in. At the moment, it contains just stubs.
- `XMLTypes.hs` contains the type definitions for our simplified XML trees.

- `Play.hs` contains a sample XML value. To avoid getting into details of parsing actual XML concrete syntax, we'll work with just this one value for purposes of this assignment.

  The XML value in `Play.hs` has the following structure (in standard XML syntax):

```
<PLAY>
   <TITLE>TITLE OF THE PLAY</TITLE>

   <PERSONAE>
     <PERSONA> PERSON1 </PERSONA>
     <PERSONA> PERSON2 </PERSONA>
     ... -- MORE PERSONAE
   </PERSONAE>

   <ACT>
      <TITLE>TITLE OF FIRST ACT</TITLE>
      <SCENE>
          <TITLE>TITLE OF FIRST SCENE</TITLE>
          <SPEECH>
              <SPEAKER> PERSON1 </SPEAKER>
              <LINE>LINE1</LINE>
              <LINE>LINE2</LINE>
              ... -- MORE LINES
          </SPEECH>
          ... -- MORE SPEECHES
      </SCENE>
      ... -- MORE SCENES
   </ACT>

   ... -- MORE ACTS
</PLAY>
```

- `sample.html` contains a (very basic) HTML rendition of the same information as `Play.hs`. You may want to have a look at it in your favorite browser.

  The HTML in `sample.html` has the following structure (with whitespace added for readability).

```
<html>
  <body>
    <h1>TITLE OF THE PLAY</h1>
    <h2>Dramatis Personae</h2>
    PERSON1<br/>
    PERSON2<br/>
    ...
    <h2>TITLE OF THE FIRST ACT</h2>
    <h3>TITLE OF THE FIRST SCENE</h3>
    <b>PERSON1</b><br/>
    LINE1<br/>
    LINE2<br/>
    ...
    <b>PERSON2</b><br/>
    LINE1<br/>
    LINE2<br/>
```

```
    ...
    <h3>TITLE OF THE SECOND SCENE</h3>
    <b>PERSON3</b><br/>
    LINE1<br/>
    LINE2<br/>
    ...
  </body>
</html>
```

Main assignment

Write a function `playToHtml` that converts an XML structure representing a play to another XML structure that, when printed, yields the HTML specified above (but with no whitespace except what's in the textual data in the original XML).

The `main` action that we've provided will use your function to generate a file `dream.html` from the sample play. The contents of this file after your program runs must be *character for character identical* to `sample.html`. Our `main` action tests this.

Important: The purpose of this assignment is *not* just to "get the job done"—i.e., to produce the right HTML. A more important goal is to think about what is a *good* way to do this job, and jobs like it. To this end, your solution should be organized into two parts:

1. a collection of *generic* functions for transforming XML structures that have nothing to do with plays, plus
2. a *short* piece of code (a single definition or a collection of short definitions) that uses the generic functions to do the particular job of transforming a play into HTML.

Obviously, there are many ways to do the first part. The main challenge of the assignment is to find a clean design that matches the needs of the second part.

You will be graded not only on correctness (producing the required output), but also on the elegance of your solution and the clarity and readability of your code and documentation. Style counts.

It is strongly recommended that you rewrite this part of the assignment a couple of times: get something working, then step back and see if there is anything you can abstract out or generalize, rewrite it, then leave it alone for a few hours or overnight and rewrite it again. Try to use some of the higher-order programming techniques we've been discussing in class.

Submission instructions

- Include your solutions to the warm-up exercises in `Mine.hs`.
- Make sure that compiling and running `Main` prints `Success`.
- Make sure `Mine.hs` is accepted by GHC without errors or warnings and follows the other rules in the Style Guide on the course web page. (You can ignore the "TS" requirement — the `Main` program *is* the testing code in this assignment.)
- Put your name in a comment at the top of the file.
- Email just the file `Mine.hs` to both `jschorr@seas.upenn.edu` and `bcpierce@cis.upenn.edu`.