Advanced Programming (CSE 399)
Homework Assignment 5

Due Friday, February 18, at noon

A source file `Main.hs` that forms the starting point of this assignment is available on the Schedule page under the main course web page. Begin by grabbing this file and making sure that you can successfully run it (using either Hugs or GHC). The testing function at the bottom of `Main.hs` gives examples of the expected behaviors of your solutions to the various parts of the assignment.

1. `Main.hs` includes the parser for arithmetic expressions that was presented in class.

   Modify the expression parser so that it accepts negative integers (like `-5`).

2. Further modify the expression parser so that it can handle *exponentiation* expressions like `2^4` (which evaluates to 16). Make sure the precedence of exponentiation is higher than that of multiplication or division, so that `3*2^2` evaluates to 12. To achieve this, you will need to introduce a new syntactic category along the lines of `term` and `factor`.

3. Further modify the expression parser so that it will accept arbitrary amounts of whitespace (space, tab, or newline characters) between tokens.

4. Implement a parser for XML data.

   Use the same `XML` datatype as in the last assignment (already provided in `Main.hs`).

```
data XML =
         PCDATA  String
       | Element ElementName [Attribute] [XML]

type ElementName = String
type Attribute = (String,String)
```

   For purposes of this assignment, use the following simplified rules for XML concrete syntax:

   - a *PCDATA item* consists of a non-empty sequence of characters not containing `<` or `>`;
   - an *identifier* is a non-empty sequence of alphanumeric characters;
   - a *quoted string* consists of the character `"`, any sequence of characters not including `"`, and the character `"`;
   - an *attribute* consists of an identifier, the character `=`, and a quoted string;
   - an *opening tag* consists of the character `<`, an identifier, a (possibly empty) sequence of attributes, and the character `>`;
   - a *closing tag* consists of the character `<`, the character `/`, an identifier, and the character `>`;
   - an *element* consists of an opening tag, a (possibly empty) sequence of either PCDATA or Elements, and a (matching) closing tag.

5. **Extra credit:** Modify the `Parser` monad and its associated primitive parsers so that, when parsing fails, it yields an error message reporting what went wrong. For example, the type of possible results of parsing might be changed to

```
data ParseResult = Good  [(XML,String)]
                 | Error String
```

and doing

```
parse xml "<tag<"
```

might yield something like this:

```
Error "expected > but found <"
```

(This specification is intentionally vague. The details are up to you. :-)

6. **Extra extra credit:** Further modify the `Parser` monad so that, when parsing fails, the `Error` result includes not only a descriptive string but also the character position in the input where the error was detected.

Submission instructions:

- Put the solutions to all parts into the file `Main.hs` and email this file to `bcpierce@cis.upenn.edu`.

  If you choose to do the extra credit part(s), put this in a separate file along with some testing code that shows how it works and send it *as a separate email message* including the words "extra credit" in the subject line.