**SXML Reformat**

In this assignment, you will write a program to reformat SXML (simplified XML) documents. The program will use the `Parser` module to parse an SXML document and will use the `PrettyPrint` module to pretty print the SXML document.

Download `Parser.hs`, which defines a type and combinators for parsing, `PrettyPrint.hs`, which defines a type and combinators for pretty printing, and `SXMLReformat.hs`, which defines a type for (simplified) XML documents.

The `SXML` type is defined as follows:

```haskell
newtype SXML = SXML Elt deriving (Eq, Read, Show)

data Elt = Elt String [Att] [Item] deriving (Eq, Read, Show)
-- Invariant:: forall (Elt n atts items) . validName n

data Att = Att String String deriving (Eq, Read, Show)
-- Invariant:: forall (Att n v) . validName n && validAttValue v

data Item = IElt Elt | IText String deriving (Eq, Read, Show)
-- Invariant:: forall (IText s) . validText s
```

An (internal or logical) SXML document (`SXML`) is a single “root” element (`Elt`). An element (`Elt`) is comprised of a name (`String`), a (possibly empty) list of attributes (`[Att]`), and a (possibly empty) list of items (`[Item]`). An attribute (`Att`) is comprised of a name (`String`) and an attribute value (`String`). An item (`Item`) is either an element (`IElt Elt`) or text (`IText String`).

Note the following invariants for names, attribute values, and text:

- A name is a non-empty sequence of characters; the first character must be either `:`, `.` or a letter; for each of the remaining characters, it must be either `:`, `.` or a letter, or a digit.

```haskell
validName :: String -> Bool
validName [] = False
validName (c:cs) = nameStartChar c && all nameChar cs
  where nameStartChar = (== ':' ||| (== '.') ||| isAlpha
      nameChar = nameStartChar ||| (== '-') ||| (== .'') ||| isDigit
```

- An attribute value is a sequence of characters; no character may be either `<`, `'>`, or `'"'; if a character is a white-space character, then it must be `'`

```haskell
validAttValue :: String -> Bool
validAttValue = all ((/= '<') &&& (/= '>') &&& (/= '"') &&& (not . isSpace))
```

- Text is a non-empty sequence of characters; no character may be either `<`, `'>`, or a white-space character.

```haskell
validText :: String -> Bool
validText = (not . null) &&& all ((/= '<') &&& (/= '>')) &&& (not . isSpace)
```
Parse

An (external or physical) SXML document is a sequence of characters obeying a number of well-formedness rules. Figure 1 shows a sample SXML document.

The following is an informal description of SXML documents:

- An SXML document (SXML) is a single “root” element (Elt) preceeded and followed by any number of miscellaneous (Misc).
- A miscellaneous (Misc) is either a comment (Comment) or a non-empty sequence of white-space characters.
- A comment (Comment) is the ‘<!--’ characters, followed by any number of characters that does not include a ‘--’ character immediately followed by another ‘--’ character, followed by the ‘-->’ characters.
- An element (Elt) is either an empty-tag element (EmptyTagElt) or a start-end-tag element (StartEndTagElt).
- An empty-tag element (EmptyTagElt) is the ‘<’ character, followed by a name (Name), followed by any number of attributes (Att) each preceeded by a non-empty sequence of white-space characters, followed by any number of white-space characters, followed by the ‘/>’ characters. Note: there must be no white-space characters between the ‘<’ character and the name. Note: there must be white-space characters between the name and the first attribute and between attributes. Note: there may be white-space characters before the ‘/>’ characters.
- A start-end-tag element (StartEndTagElt) is a start tag (StartTag), followed by any number of items (Item) each preceeded by any number of miscellaneous (Misc), followed by any number of miscellaneous (Misc), followed by an end tag (EndTag). The name of the start tag must be the same as the name of the end tag.
- A start-tag (StartTag) is the ‘<’ character, followed by a name (Name), followed by any number of attributes (Att) each preceeded by a non-empty sequence of white-space characters, followed by any number of white-space characters, followed by the ‘>’ character. Note: there must be no white-space characters between the ‘<’ character and the name. Note: there must be white-space characters between the name and the first attribute and between attributes. Note: there may be white-space characters before the ‘>’ character.
- An end-tag (EndTag) is the ‘</’ characters, followed by a name (Name), followed by any number of white-space characters, followed by the ‘>’ character. Note: there must be no white-space characters between the ‘</’ characters and the name. Note: there may be white-space characters before the ‘>’ character.
- A name (Name) is a non-empty sequence of characters; the first character must be either ‘:’, ‘_’, or a letter; for each of the remaining characters, it must be either ‘:’, ‘_’, ‘=’, ‘.’, a letter, or a digit. (This is the same requirement as for the Elt and Att constructors above.)
- An attribute (Att) is a name, followed by the ‘=’ character, followed by an attribute value (AttValue), followed by the ‘>’ character. Note: there must be no white-space characters between the name and the ‘=’ character and there must be no white-space characters between the ‘=’ character and the initial ‘>’ character.
- An attribute value (AttValue) is a sequence of characters; no character may be either ‘<’, ‘>’, or ‘”; if a character is a white-space character, then it must be ‘ ’. (This is the same requirement as for the Att constructor above.)
- An item (Item) is either an element (Elt) or text (Text).
- Text (Text) is a non-empty sequence of characters; no character may be either ‘<’, ‘>’, or a white-space character. (This is the same requirement as for the IText constructor above.)

Figure 2 is a more formal description of SXML documents as a grammar using Extended Backus-Naur-Form (EBNF). In this notation, $X^+$ means any number (zero or more) of $X$s; $X^*$ means a non-empty sequence (one or more) of $X$s.

Note that when parsing an external SXML document to an internal SXML document, all comments and white-space characters are discarded (with the exception of ‘ ’ (space) characters in attribute values).
Figure 1: Sample SXML document
Simplified XML

\[
\begin{align*}
SXML & \rightarrow \text{Misc}^* \text{Elt Misc}^* \\
\text{Misc} & \rightarrow \text{Comment} \\
\text{Misc} & \rightarrow \text{Space}^+ \\
\text{Comment} & \rightarrow \langle!|-- (\text{[}]) | (\text{[}])\rangle^* \rightarrow (1) \\
\text{Elt} & \rightarrow \text{EmptyTagElt} \\
\text{Elt} & \rightarrow \text{StartEndTagElt} \\
\text{EmptyTagElt} & \rightarrow <\text{Name} (\text{Space}^+ \text{Att})^* \text{Space}^*/> \\
\text{StartEndTagElt} & \rightarrow \text{StartTag (Misc}^* \text{Item})^* \text{Misc}^* \text{EndTag} (2) \\
\text{StartTag} & \rightarrow <\text{Name} (\text{Space}^+ \text{Att})^* \text{Space}^*/> \\
\text{EndTag} & \rightarrow </\text{Name} \text{Space}^*/> \\
\text{Name} & \rightarrow \ldots (3) \\
\text{Att} & \rightarrow \text{Name} = "\text{AttValue}" \\
\text{AttValue} & \rightarrow \ldots (4) \\
\text{Item} & \rightarrow \text{Elt} \\
\text{Item} & \rightarrow \text{Text} \\
\text{Text} & \rightarrow \ldots (5)
\end{align*}
\]

(1): "((\text{[}]) | (\text{[}])\rangle)^*" is a regular expression denoting any number of characters that does not include a ‘–’ character immediately followed by another ‘–’ character.

(2): The Name of the StartTag must match the Name of the EndTag.

(3): Name is a non-empty sequence of characters; the first character must be either ‘:’, ‘_’, or a letter; for each of the remaining characters, it must be either ‘:’, ‘_’, ‘-’, ‘.’, a letter, or a digit.

(4): AttValue is a sequence of characters; no character may be either ‘<’, ‘>’, or ‘”’; if a character is a white-space character, then it must be ‘ ’ (space).

(5): Text is a non-empty sequence of characters; no character may be either ‘<’, ‘>’, or a white-space character.

Figure 2: SXML document grammar in EBNF
1. (30pts) Define a parser
   
   - sxmlP :: Parser SXML

   such that sxmlP parses an SXML document.

   Examples:
   
   - parseMaybe sxmlP "<root></root>" \rightarrow Just (SXML (Elt "root" [] []))
   - parseMaybe sxmlP "<root/>" \rightarrow Just (SXML (Elt "root" [] []))
   - parseMaybe sxmlP "<root title="sample"> Some <b>bold</b> text. </root>"
     \rightarrow Just (SXML (Elt "root"
     [Att "title" "sample"]
     [IText "Some",IElt (Elt "b" [] [IText "bold"]),IText "text."]))
   - parseMaybe sxmlP "<root title="sample"> Some <b>bold</b> text. </toor>" \rightarrow Nothing

   Notes:
   
   - An (external or physical) SXML document should parse to exactly one (internal or logical) SXML document. That is, for any string s, parseList sxmlP s should return either an empty list or a singleton list.

   Hints:
   
   - Define helper parsers, including, but not limited to,
     
     - miscP :: Parser ()
     - commentP :: Parser ()
     - eltP :: Parser Elt
     - nameP :: Parser String
     - attP :: Parser Att
     - attValueP :: Parser String
     - itemP :: Parser Item
     - textP :: Parser String

     such that each parses the appropriate fragment of an SXML document. Note that the miscP and commentP parsers return (), since such fragments do not contribute to an internal SXML document (SXML).

   - Be sure to test each parser; in particular, you will need to test the sxmlP parser on many more examples than those given above.
Pretty Print

Figures 3, 4, 5, 6, and 7 show the sample SXML document from Figure 1 pretty printed for various page widths.

The following are some requirements for pretty printing SXML documents:

- A pretty printed (internal or logical) SXML document must be an (external or physical) SXML document. That is, a pretty printed SXML document must obey the well-formedness rules of the previous section.
- An element with no items must be printed as an empty-tag element.
- There must be either one space character or a line break between items.

The following are some additional guidelines for pretty printing SXML documents:

- When the attributes of an element cannot all be printed on the same line, the attributes should be indented relative to the start-tag name.
- When the items of an element cannot be printed on the same line as the start tag, the items should be indented relative to the start tag.
- Sequences of text items should fill each line with as much text as will fit before starting a new line.
- The overall layout should be “pretty”.

Pretty Print

Figures 3, 4, 5, 6, and 7 show the sample SXML document from Figure 1 pretty printed for various page widths.

The following are some requirements for pretty printing SXML documents:

- A pretty printed (internal or logical) SXML document must be an (external or physical) SXML document. That is, a pretty printed SXML document must obey the well-formedness rules of the previous section.
- An element with no items must be printed as an empty-tag element.
- There must be either one space character or a line break between items.

The following are some additional guidelines for pretty printing SXML documents:

- When the attributes of an element cannot all be printed on the same line, the attributes should be indented relative to the start-tag name.
- When the items of an element cannot be printed on the same line as the start tag, the items should be indented relative to the start tag.
- Sequences of text items should fill each line with as much text as will fit before starting a new line.
- The overall layout should be “pretty”.

Pretty Print

Figures 3, 4, 5, 6, and 7 show the sample SXML document from Figure 1 pretty printed for various page widths.

The following are some requirements for pretty printing SXML documents:

- A pretty printed (internal or logical) SXML document must be an (external or physical) SXML document. That is, a pretty printed SXML document must obey the well-formedness rules of the previous section.
- An element with no items must be printed as an empty-tag element.
- There must be either one space character or a line break between items.

The following are some additional guidelines for pretty printing SXML documents:

- When the attributes of an element cannot all be printed on the same line, the attributes should be indented relative to the start-tag name.
- When the items of an element cannot be printed on the same line as the start tag, the items should be indented relative to the start tag.
- Sequences of text items should fill each line with as much text as will fit before starting a new line.
- The overall layout should be “pretty”.

Pretty Print

Figures 3, 4, 5, 6, and 7 show the sample SXML document from Figure 1 pretty printed for various page widths.

The following are some requirements for pretty printing SXML documents:

- A pretty printed (internal or logical) SXML document must be an (external or physical) SXML document. That is, a pretty printed SXML document must obey the well-formedness rules of the previous section.
- An element with no items must be printed as an empty-tag element.
- There must be either one space character or a line break between items.

The following are some additional guidelines for pretty printing SXML documents:

- When the attributes of an element cannot all be printed on the same line, the attributes should be indented relative to the start-tag name.
- When the items of an element cannot be printed on the same line as the start tag, the items should be indented relative to the start tag.
- Sequences of text items should fill each line with as much text as will fit before starting a new line.
- The overall layout should be “pretty”.

Pretty Print

Figures 3, 4, 5, 6, and 7 show the sample SXML document from Figure 1 pretty printed for various page widths.

The following are some requirements for pretty printing SXML documents:

- A pretty printed (internal or logical) SXML document must be an (external or physical) SXML document. That is, a pretty printed SXML document must obey the well-formedness rules of the previous section.
- An element with no items must be printed as an empty-tag element.
- There must be either one space character or a line break between items.

The following are some additional guidelines for pretty printing SXML documents:

- When the attributes of an element cannot all be printed on the same line, the attributes should be indented relative to the start-tag name.
- When the items of an element cannot be printed on the same line as the start tag, the items should be indented relative to the start tag.
- Sequences of text items should fill each line with as much text as will fit before starting a new line.
- The overall layout should be “pretty”.

Pretty Print

Figures 3, 4, 5, 6, and 7 show the sample SXML document from Figure 1 pretty printed for various page widths.

The following are some requirements for pretty printing SXML documents:

- A pretty printed (internal or logical) SXML document must be an (external or physical) SXML document. That is, a pretty printed SXML document must obey the well-formedness rules of the previous section.
- An element with no items must be printed as an empty-tag element.
- There must be either one space character or a line break between items.

The following are some additional guidelines for pretty printing SXML documents:

- When the attributes of an element cannot all be printed on the same line, the attributes should be indented relative to the start-tag name.
- When the items of an element cannot be printed on the same line as the start tag, the items should be indented relative to the start tag.
- Sequences of text items should fill each line with as much text as will fit before starting a new line.
- The overall layout should be “pretty”.
<document>
  <section number="1">
    <p color="red">This paragraph is <b>red</b>.</p><p size="20">This paragraph is <b>big</b>.</p><p font="Times" color="red" size="20" align="center">This paragraph is <b>quite fancy</b>.</p>
    <ul>
      <li>List 1</li><li>List 2</li>
      <li>List 3
        <ul>
          <li>List a</li><li>List b</li><li>List c</li>
        </ul>
      </li>
      <li>List 4</li>
    </ul>
    <br/>
    <img src="logo.png" alt="logo"/>
  </section>
</document>

Figure 5: Sample SXML document pretty printed for page width 60

<document>
  <section number="1">
    <p color="red">This paragraph is <b>red</b>.</p><p size="20">This paragraph is <b>big</b>.</p><p font="Times" color="red" size="20" align="center">This paragraph is <b>quite fancy</b>.</p>
    <ul>
      <li>List 1</li><li>List 2</li>
      <li>List 3
        <ul>
          <li>List a</li><li>List b</li><li>List c</li>
        </ul>
      </li>
      <li>List 4</li>
    </ul>
    <br/>
    <img src="logo.png" alt="logo"/>
  </section>
</document>

Figure 6: Sample SXML document pretty printed for page width 40
<document>
  <section number="1">
    <p color="red">
      This paragraph is <b>red</b>.
    </p>
    <p size="20">
      This paragraph is <b>big</b>.
    </p>
    <p font="Times" color="red" size="20" align="center">
      This paragraph is <b>quite fancy</b>.
    </p>
    <ul>
      <li> List 1</li>
      <li> List 2</li>
      <li> List 3
        <ul>
          <li> List a</li>
          <li> List b</li>
          <li> List c</li>
        </ul>
      </li>
      <li> List 4</li>
    </ul>
    <br/>
    <img src="logo.png" alt="logo"/>
  </section>
</document>

Figure 7: Sample SXML document pretty printed for page width 20
2. (15pts) Define a function

- `sxmlD :: SXML -> Doc`

such that `sxmlD sxml` returns a “pretty” document representing the SXML document `sxml`.

Notes:
- `sxmlD` must follow the requirements given above.
- `sxmlD` might follow the guidelines given above.
- `sxmlD` does not need to match the output in Figures 3, 4, 5, 6, and 7 exactly.

Hints:
- Define helper pretty printers, including, but not limited to,
  - `eltD :: Elt -> Doc`
  - `attD :: Att -> Doc`
  - `itemsD :: [Item] -> Doc`

such that each pretty prints the appropriate fragment of an SXML document.
- Experiment!

3. (5pts) In a comment, describe your `sxmlD`. That is, give a description of the pretty-printed SXML documents produced by `sxmlD`. Comment on any design decisions that you made in developing `sxmlD`. Comment on any “ugly” aspects that remain in pretty-printed SXML documents produced by `sxmlD`. 
Program

4. (10pts) Define an I/O action

- main :: IO ()

that, when run, parses the (external or physical) SXML document on standard input to an (internal or logical) SXML document, pretty prints the (internal or logical) SXML document for a specified page width to an (external or physical) SXML document on standard output. The page width is specified via the first command-line argument; if there are no command-line arguments, or if the first command-line argument cannot read as an Int, or if the first-command line argument is a negative integer, then pretty print the SXML document for page width 80. If the standard input cannot be parsed as an SXML document, then print “** PARSE ERROR **” on standard output.

Hints:

- Useful Prelude types and functions include:
  - getContents :: IO String
- Useful System.Environment functions include:
  - getArgs :: IO [String]
- Useful Text.Read functions include:
  - readMaybe :: Read a => String -> Maybe a

See Figure 8 for sample interactions.

A reference solution executable on queeg.cs.rit.edu or the Linux ICL machines is available on the CS Department file system at:

/usr/local/pub/mtf/psfp-20191/homework08/SXMLReformat

Examples

5. (5pts) In a text file named good.sxml, develop an interesting example SXML document that fully exercises all aspects of your parser and pretty printer.

Note:

- When your good.sxml is reformatted by the reference solution executable for page width 40, the output SXML document must have a minimum of 30 lines.
- Simply using or modifying one of the provided sample SXML documents will receive no credit.

6. (5pts) In a text file named bad.sxml, make a (small) change to your good.sxml such that bad.sxml cannot be parsed as an SXML document (by either your SXMLReformat.hs or the reference solution executable).

Requirements and Submission

Your submission must :load into ghci without errors and must execute with runhaskell without errors; submissions that have parse errors or type errors will receive no credit.

Submit SXMLReformat.hs, good.sxml, and, bad.sxml to the Homework08 Assignment on MyCourses by the due date.
Figure 8: main examples