The University of Hong Kong

COMP3258: Functional Programming

Assignment 2 (Parsing)

Deadline: 23:55, Nov 15, 2019 (HKT)

---

1 Background

Assume you have a set of boxes.

Each box has its own attributes: length, width and height. A box could be placed inside another box as long as the inner one has decreasing length, width and height.

A box can contains as many as small boxes if the sum of length of these boxes is less than length of the outer box, if the maximum of width of these boxes is less than width of the outer box and if the maximum of height of these boxes is less than height of the outer box.

Let me describe these boxes in Haskell. Assume you have data structure:

```haskell
data Box = Box Int Int Int [Box] deriving (Eq)
```

The first parameter represents the length of a box, the second parameter represents the width of a box, the third parameter represents the height of a box and the last parameter tells us if there are inner boxes.

Problem 1. (10 pts) Implement `numBox :: Box -> Int`, computing how many boxes you have. You can assume the input is always valid for this question.

Sample inputs and outputs:

```
ghci> a = Box 1 2 3 []
ghci> b = Box 1 2 3 []
ghci> c = Box 3 5 7 [a, b]
ghci> numBox a
1
ghic> numBox c
3
```

Problem 2. (10 pts) The output of built-in `show` function doesn't look pretty when you try to inspect the data structure of boxes. Re-implement `show` function, printing these boxes in a pretty format. More precisely, you should derive a `Show` instance like

```haskell
instance Show Box where
    show _ = -- your code here
```

Rules: If there is no box placed inside a box, your function should return `(length:%d, width:%d, height:%d)`. Here, `%d` represents an integer. If there are other boxes placed inside a box, your function should return `(length:%d, width:%d, height:%d, %s)`. Here, `%s` represents the formatted information of inner boxes. Two boxes placed next to each other are separated by a comma and a space.

You can assume the input is always valid for this question.

Sample inputs and outputs:

```
ghci> a
(length:1, width:2, height:3)
ghic> c
(length:3, width:5, height:7 (length:1, width:2, height:3), (
length:1, width:2, height:3))
```

2 Parser

In this task, the information of boxes will be stored in a string written by a formatted XML-like language.

A legal example of XML-like language looks like:

```
<box length=3 width=5 height=7>
    <box length=1 width=2 height=3 >
    </box>
    <box  length=1 width=2 height=3>
    </box>
</box>
```

Let us describe this XML-like language through Backus–Naur Form:

```
<whitespace>     ::= '\t' | ' ' | '\n' | '\r'
<opt-whitespace> ::= <whitespace> <opt-whitespace> | <whitespace>
<digit>          ::= "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"
<digits>         ::= <digit> | <digit> <digits>
<boxes>          ::= "<box" <opt-whitespace>
    "length" '=' <digits> <opt-whitespace>
    "width"  '=' <digits> <opt-whitespace>
    "height" '=' <digits> <opt-whitespace> '>'
    <opt-whitespace> <box_list> <opt-whitespace> "</box>"
<box_list> := "" | <boxes> <opt-whitespace> <box_list>
```

Note:

- The order of attributes is important.
- Input with correct syntax doesn't means it is realistic.
- Valid input implies syntax correct and realistic.
- Although leading zero is allowed in grammar, your function should ignore that when you store the number in an integer.

Problem 3. (60 pts) Implement function `parse :: String -> Box`, converting an input with XML-like language format to our pre-defined Haskell data structure. You can assume the input is always valid for this question.

Sample inputs and outputs:

```
ghci> test1 = "<box length=10 width=5 height=3></box>"
ghci> parse test1
(length:10, width:5, height:3)
ghci> test2 = "<box length=10 width=5 height=3>
\n\t<box length=8 width=2 height=2>\n\t</box>\n</box>"
ghci> parse test2
(length:10, width:5, height:3 (length:8, width:2, height:2))
ghci> test3 = unlines ["<box length=10 width=5 height=3>",
"<box length=8 width=4 height=1></box>",
"<box length=1 width=4 height=1></box>","</box>"]
ghci> parse test3
(length:10, width:5, height:3 (length:8, width:4, height:1
), (length:1, width:4, height:1))
```

Note: Haskell doesn't support multi-line string constant, thus if needed you should explicitly write the line break (`\n` or `\r`) or use `unlines` function.

3 Validation

However, your algorithm might be fed an illegal input.

Problem 4. (20 pts) Implement function `isValid :: String -> Bool`, checking whether an input with XML-like language format is valid.

```
ghci> isValid test1
True
ghci> isValid test2
True
ghci> isValid test3
True
```

There are two sorts of invalid inputs:

1) The string violates the XML-like grammar specification.

Below shows some incorrect inputs:

- The type of size is not an integer.
- he order of attributes is wrong.
- Lack of attribute.
- Typo.
- Other grammar errors.

```
ghci> err1 = "<box length=3 width=bug height=50></box>"
ghci> err2 = "<box width=3 length=2 height=50></box>"
ghci> err3 = "<box length=3 width=1></box>"
ghci> err4 = "<box length=3 width=4 heigh=50></box>"
ghci> err5 = "<box length=3 width=2 height=50>"
ghci> isValid err1
False
ghci> isValid err2
```

```
False
ghci> isValid err3
False
ghci> isValid err4
False
ghci> isValid err5
False
```

2) The string is syntactically correct but violates the placement rules.

Our placement rule says: A box can contains as many as small boxes if

- The sum of length of these boxes is less than length of the outer box
- The maximum of width of these boxes is less than width of the outer box
- The maximum of height of these boxes is less than height of the outer box

```
ghci> err6 = "<box length=3 width=4 height=5><box length=3 width=3
height=4></box></box>"
ghci> err7 = "<box length=3 width=4 height=5><box length=2 width=4
height=4></box></box>"
ghci> err8 = "<box length=3 width=4 height=5><box length=2 width=4
height=4></box><box length=1 width=4 height=4></box></box>"
ghci> isValid err6
False
ghci> isValid err7
False
ghci> isValid err8
False
```

4 Policies

Constraints

- You can ONLY use functions from these libraries:
  - Prelude
  - Data.Char
  - Data.List
  - Control.Monad
  - Control.Applicative
- GHC extension is not allowed.

If you break the constraints, you won't get any credits.

Grading

Your code will be graded automatically by a grading program.

Please make sure auto-grader can find your function and there is no grammar error in your code.

Please make sure no statement like `module XXX where` exists in your file, otherwise there will be 5 pts deduction.

If there is question you don't implement, remove it from your code.

For question 1, there are 2 test cases. Each test case is worth 5 points.

For question 2, there are 2 test cases. Each test case is worth 5 points.

For question 3, there are 6 test cases. Each test case is worth 10 points.

For question 4, there are 8 test cases. These 8 test cases will be divided into 4 groups. In each groups, a test case has a valid input and the other test case has an invalid output. If you can pass two test cases in one group you will get 5 points.

The correctness of your implement of question 2 won't affect the grading of your implement of question 3.

Time limit for each test case: 1 sec.

Only does your code have same output as standard output then your code is considered as passing the test case.

No partial points exists for each test case.

Code style and submission (5 pts.)

All functions should be implemented in a single Haskell file, named as A2_XXX.hs, with XXX replaced by your UID. Your code should be well-written (e.g. proper indentation, names, and type annotations) and documented. Please submit your solution on Moodle before the deadline.