universität innsbruck

Functional Programming — WS 2021 — LVA 703025

Exercise Sheet 2, 10 points — Deadline: Wednesday, October 20, 2021, 6am

- Mark your completed exercises in the OLAT course of the PS.
- Upload your .hs-file of Exercise 3 in OLAT.
- Your .hs-file should be compilable with ghci.

Exercise 1 *Typing* — 4 p.

Given the definition

```
plus1 x = x + 1
```

which of the following typing judgments are valid? Justify your answers.

1. `0 :: Bool` (1 point)
2. `head "test" :: Char` (1 point)
3. `'hello' :: String` (1 point)
4. `plus1 :: Integer -> Integer` (1 point)

Solution 1

1. Since `0` is a number but type `Bool` consists of the truth-values `True` and `False`, the typing judgment `0 :: Bool` is not valid.
2. The expression `head "test"` extracts the first character of the string `"test"`, which is `'t'` and has type `Char`. Therefore, the typing judgment `head "test" :: Char` is valid.
3. In Haskell strings are written between double quotes (`"`) and not single quotes (`'`). Therefore, this expression is not even syntactically correct and thus the typing judgment `'hello' :: Char` is invalid.
4. In Haskell, addition of `Integers` is the function `(+) :: Integer -> Integer -> Integer`. That is, a function taking two `Integers` as arguments and delivering an `Integer` as result. Inside the definition of `plus1` the second argument of `(+)` is fixed to be `1 :: Integer`. Therefore, the resulting function `plus1` takes a single argument of type `Integer` and delivers an `Integer` as result. Thus, the typing judgment `plus1 :: Integer -> Integer` is valid.

Exercise 2 *Parsing expressions* — 3 p.

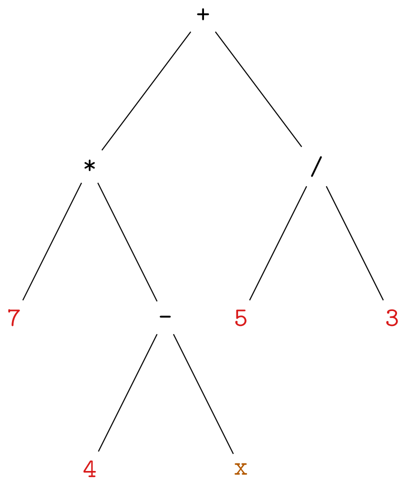
Draw the abstract syntax trees of the following expressions:

1. `7 * (4 - x) + 5 / 3` (1 point)
2. `(x < 10) || (y > 15)` (1 point)
3. `average 5 10 * square 2 + 10` (1 point)

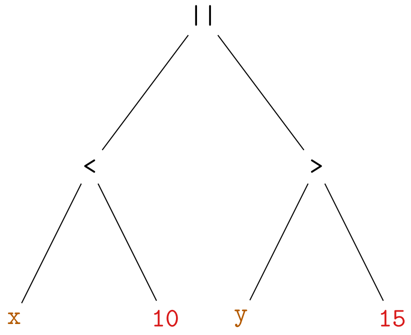
Remark: function applications (e.g., `square 7`) bind stronger than operator applications (e.g., `3 * 4`).

Solution 2

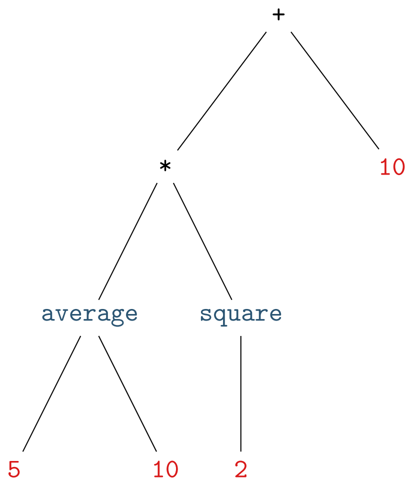
1. `7 * (4 - x) + 5 / 3`



2. `(x < 10) || (y > 15)`



3. `average 5 10 * square 2 + 10`



Exercise 3 *Modelling* **3 p.**

In graphical user interfaces (GUIs) a *menu* typically consists of *items* and submenus. One specific application of such menus is website navigation, where items would consist of a label (the text to click on) and a link (the URL of the website to navigate to when the item is clicked).

1. Give a Haskell datatype definition to model items for website navigation as described above. Moreover, define constants that represent the items OLAT (`https://lms.uibk.ac.at`) and FP (`http://cl-informatik.uibk.ac.at/teaching/ws21/fp`) (1 point)

2. Give a Haskell datatype definition to model menus that may contain up to two items. Moreover, define a constant that represents a menu containing the items for OLAT and FP from above. (1 point)
3. Change your definition from the previous exercise such that a menu may contain an arbitrary number of items. Moreover, define a constant that represents a menu with at least three items and also represent this constant as a tree as shown on the slides of week 2, page 23. (1 point)

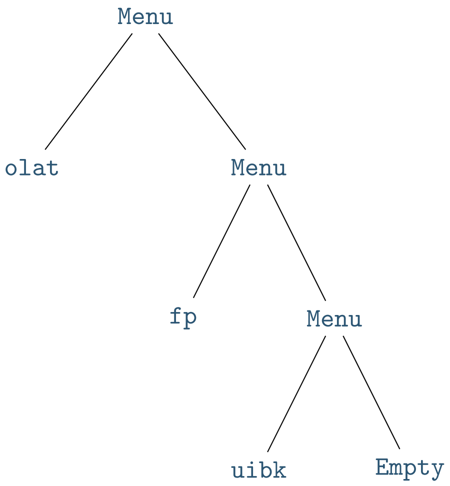
Solution 3

1.
```haskell
data Item = Item String String

olat = Item "OLAT" "https://lms.uibk.ac.at"
fp = Item "FP" "http://cl-informatik.uibk.ac.at/teaching/ws21/fp"
```
2.
```haskell
data Menu2 = Menu2Empty         -- the empty menu
           | Menu2One Item      -- a menu with a single item
           | Menu2Two Item Item -- a menu with two items
  deriving Show

menu2 = Menu2Two olat fp
```
3.
```haskell
  data MenuList = Empty | Menu Item MenuList deriving Show

  uibk = Item "UIBK" "https://www.uibk.ac.at"
  menuList = Menu olat
               (Menu fp
                 (Menu uibk Empty)
               )
```
   - Tree representation


Menu
olat
Menu
fp
Menu
uibk
Empty