universität
innsbruck

Functional Programming | WS 2020 | LVA 703025

Exercise Sheet 2, 10 points | Deadline: Wednesday, October 21, 2020, 6am

- Upload your solution(s) for exercise 2.1 into a single pdf (as it contains drawings) in OLAT.
- Upload the .hs-file for exercises 2.2 and 2.3 in OLAT. Please upload your solutions in a single .hs-file. You can use the template .hs-file that is provided on the proseminar page. The .hs-file must compile with ghci.
- Don't forget to mark your completed exercises in OLAT.

Exercise 2.1 *Parsing expressions* 4 p.

For the following expressions, determine whether parsing succeeds or fails according to the parsing algorithm described in the lecture, see slides 2/16–23. If it succeeds, draw the abstract syntax tree. If it fails, explain in which way(s) parentheses could be added to make parsing succeed.

1. `6 - (5 - (4 + (3 * 2)))`
2. `6 - 5 - 4 + 3 * 2`
3. `x * f (f 5 6)`
4. `(3 > 5) == 6 < 7`

Exercise 2.2 *If-Then-Else* 3 p.

Haskell provides the `if-then-else` expression that allows us to branch evaluation based on a boolean condition, cf. also slide 2/24 which will be presented on Monday in the lecture. For example, `if 5 < 6 then 7 * 8 else 9 * 9` evaluates to `56`. Here are the two evaluation rules for `if-then-else` expressions:

```
if True  then t else e = t
if False then t else e = e
```

These rules state that in order to evaluate an `if-then-else` expression, we first fully evaluate the boolean condition after the `if` keyword and then choose either the `then` branch or the `else` branch accordingly.

1. Evaluate the following two expressions fully and step by step on paper to get a better understanding of boolean expressions and `if-then-else` expressions.
   - `if 5 * 8 <= 70 && 7 - 8 < 0 then 8 `div` 2 else 90`
   - `if 7 - 8 > 0 then 8 `div` 2 else (if 7 * 7 == 49 then 0 else 49)`

   (1 point)
2. Now write a Haskell function `noe` that given a number from 0 to 9, evaluates to `0` iff the (English) word for it does not contain the letter e. If the English name for the number does contain the letter e, it should return `1`. For instance, `noe 5` should return `1` since the word “five” contains an e, but `noe 4` should return `0` as “four” does not. (1 point)

3. Implement the following mathematical definition as function `arbitraryFun` in Haskell:

$$\text{arbitraryFun}(x) = \begin{cases} 8 & \text{if } x \text{ is greater than or equal to 100} \\ 19 & \text{if } x \text{ is less than 100 and divisible by 7} \\ 7 & \text{if } x \text{ is less than 100 and not divisible by 7} \end{cases}$$

   Note: The function `isDivisible` from Exercise 2.3.1 can be reused here. (1 point)

Exercise 2.3 *Defining and Reusing Functions* **3 p.**

1. Reimplent the built-in function `mod`[1] for computing the remainder of a division using only `*`, `+`, `-` and `div`. Call your function `mod1`. Using `mod1` implement a function `isDivisible` that takes two parameters `x` and `y` and returns `True` if `x` is divisible by `y` and otherwise, `False`. You may assume that no negative numbers are provided as input for both `mod1` and `isDivisible`. (1 point)

2. Write a function `stl` that returns the second to last digit of a number (in its decimal representation). If a number is only a single digit, `stl` should simply return `0`. Your function only needs to work for positive numbers.

   Examples:

   ```
   stl 18 = 1
   stl 8292 = 9
   stl 9 = 0
   ```

   Hint: You can use your implementation `mod1` or the built-in function `mod` here.

   (1 point)

3. Write a function `repl` that for a given natural number evaluates to `True` iff its *last* two digits (in its decimal representation) are the same and non-zero. For instance, `repl 1511`, `repl 666`, and `repl 123455` should all evaluate to `True`, but `repl 0`, `repl 557`, `repl 2200`, `repl 575` should all evaluate to `False`. Reuse the function `stl`. (1 point)

[1] http://zvon.org/other/haskell/Outputprelude/mod_f.html