Program Design and Data Structures, 20.0 c

Course code: 1DL201, Report code: DL201, 67%, DAG, NML,
week: 44 - 02 Semester: Autumn 2017
week: 03 - 11 Semester: Spring 2018

LAB ASSIGNMENT 10

This information is not available in English. Now showing the Swedish version.

After this lab you should be able to ...

- Construct representations of data.
- Work with enumeration types and sum types (tagged unions).
- Use the Maybe type.

Instructions

- Read the slides for Lecture 15-16 (Inductive Data Types) until slide 39 (i.e., just before Inductive Datatypes).
- Read the coding convention, specifically the section on *Datatype Representation*, to see how datatype definitions should be commented.
- Remeber to write datatype specifications according to the coding convention, with representation convention and representation invariant, for *all* datatypes that you define.

The task

PROBLEM 1

a) Design a data type for representing playing cards from a standard deck (52 cards, no jokers). Each card has one of the suits spades, hearts, diamonds or clubs, and one of the ranks 2, 3, 4, 5, 6, 7, 8, 9, 10, jack, queen, king, ace. A playing card should be represented as an element of a constructed data type, i.e., a type that has been defined with a data declaration. Document the data type.

b) Write a function greaterCard that operates on the new data type you declared for part a). The function should take two playing cards as arguments. The function should return True if the first card has a higher value than the second card, otherwise it should return False. For this exercise, ace is considered the highest rank, followed by king, queen, jack, 10, 9, ..., 2. If both cards have the same rank, suits are ordered as follows: spades > hearts > clubs > diamonds. Document the function.

c) Think about the design decisions you made when designing your data type, and how they could have been made differently. What are the advantages and disadvantages of your representation of playing cards, compared to some of the alternatives?

PROBLEM 2

Write functions for performing addition, subtraction, multiplication and division of integers. What distinguishes these functions from the normal arithmetic operations is that they should operate on the type Maybe Int rather than Int. If any argument is Nothing, the functions should return Nothing. If an operation cannot be performed (here, this only happens in the case of division by zero) the functions should also return Nothing. Otherwise, they should return the result of the arithmetic operation tagged with Just. (The point of this is that if a computation fails, program execution is not interrupted with an exception, but the result becomes Nothing.)

The four functions shall be called +?, -?, *?, and /?. You can write these functions as infix operators, just like +, - etc. Function definitions are also written in infix form: x +? y = ...

Examples of what the functions should return:

Just 1 +? Just 2 = Just 3
Just 1 -? Nothing = Nothing
Just 1 /? Just 0 = Nothing
(Just 1 /? Just 0) +? Just 2 = Nothing

OPTIONALLY: Give your operators the same precedence and fixity as the usual arithmetic operators (+, -, *, /). These properties of the standard operators was given in Lectures 3 and 6, this page describes how you declare new operators.

WHEN YOU ARE DONE

When you are done with all problems (the explorations below are optional), raise your hand or approach an assistant to have your solution graded.

If you pass this lab at least 30 minutes early and other groups are still working on it, we ask you to **help one other group**. Do not simply share your solution with them, but try to understand the (partial) solutions that they have developed so far (which may be different from yours) and the difficulties that they have. Assist them in coming up with their own solutions. Once the group that you are helping completes the lab assignment, you may leave. (If it is still early, the group that received your help should then stay to help another group.)

EXPLORATIONS

If you have finished the lab exercises and are waiting to be graded, or if you wish to explore programming in Haskell further at any time, or want to get some extra practice, have a look at the Explorations page.

Exploration problems are optional and should only be attempted after other problems are completed.
You don't have to show these answers to the lab assistants for grading.