Programming Languages — CSCE 314, Fall 2012
Assignment 1

Jaakko Järvi

September 2, 2012

General rules

Please return your answers via *CSNet*. Please submit **exactly one file**, `A1.hs`, **and nothing else**. This file should contain the definitions of the functions requested in the exercises below. `A1.hs` must compile without errors! **A file that does not compile will earn the grade zero.** If you cannot get some function to pass through the compiler, comment out the function body and leave the function body undefined—see the section *Skeleton code* for how to defined a function with an undefined body.

Remember to put your name on the top of the file.
Late penalty is 5% for every starting late day. Submissions over two days late are not accepted. Deadline is announced in class and/or on the class web pages.

A piece of advice: This assignment asks you to implement a set of Haskell functions. Each of those functions can be defined with just a few lines of code. If your implementations start to get much longer, it may be a sign that there are less complicated solutions that you may be missing.

Setup your working environment

Install GHC and a suitable editing mode for your editor, or use the GHC installed in the department's server `linux.cse.tamu.edu` or `compute.cse.tamu.edu`. Whatever your choice, get comfortable with writing, compiling, and interpreting Haskell programs.

Exercise 1: Simple Haskell drills

Implement the following functions:

```
-- Compute the nth Fibonacci number
fibonacci :: Int -> Int

-- Take the first N elements of a list
firstN :: Int -> [a] -> [a]

-- Flatten a list of lists to a single list formed by concatenation
flatten :: [[a]] -> [a]

-- Is a value an element of a list?
```

```
isElement :: Eq a => a -> [a] -> Bool

-- Remove duplicate values from a list
removeDuplicates :: Eq a => [a] -> [a]
```

Keep the naming of the functions exactly as above. The types of the functions must also be as indicated. You can of course define additional helper functions if you need them in implementing the requested functions.

The type of `isElement` function, `Eq a =>a -> [a] -> [a]`, is of the kind that we have not discussed yet. The type roughly means that for any type `a` that can be compared for equality, `isElement` has type `a -> [a] -> [a]`.

Exercise 2: More drills

In this assignment we represent mathematical *sets* with Haskell lists. Provide implementations for the following functions. Again, keep the names and types of the functions as indicated.

```
union :: Eq a => [a] -> [a] -> [a]

intersection :: Eq a => [a] -> [a] -> [a]

subset :: Eq a => [a] -> [a] -> Bool
-- subset a b is True if a is a subset of b

setEqual :: Eq a => [a] -> [a] -> Bool

powerSet :: [a] -> [[a]]
```

In all of the above functions you can assume that the incoming list arguments are sets (no duplicate values), and correspondingly, your implementations must guarantee this property for the lists they return as results.

Skeleton code

The file a1-skeleton.hs contains "stubs" for all the functions you are required to implement. Their bodies are `undefined`, a special Haskell value that has all possible types, so it can be used in any context. Also in that file you find a test suite that exercises the functions. Of course all tests fail—until you provide correct implementations for the required functions. The tests are written using the `HUnit` library. Feel free to add more tests to the test suite.

The skeleton code can be loaded to the interpreter:

```
> ghci a1-skeleton.hs
```

Evaluating the function `main` will run the tests. Alternatively, one can compile the code to an executable program, and execute it:

```
> ghc a1-skeleton.hs
> ./a1-skeleton
```