Course Project/Final Exam

Haskell for Life

`http://lacl.fr/~sivanov/doku.php?id=en:haskell_for_life` [*]

`sergiu.ivanov@lacl.fr`

Directions

This document proposes 3 problems and has the following goals:

2. test your understanding of some basic techniques used in functional programming and in Haskell, and

1. let you have some fun with Haskell.

The problems are given in no particular order; solving **any one** of the problems gives you a **passing grade**. Please include your test input files into the solutions you send me. I will send you my solutions to all of the problems *after* you have all submitted yours.

Package Management in Haskell

A standard Haskell environment (like the Haskell platform, `https://www.haskell.org/platform/`) includes a package management tool called Cabal. Running `cabal update` will download the latest list of available packages, while `cabal install library` will fetch and install `library` and all of its dependencies into your Haskell user profile. Cabal also allows creating sandboxes — isolated Haskell environments into which you can install Haskell packages without altering your Haskell user profile. Furthermore, Cabal can be used to package your own libraries and applications for subsequent distribution. The centralised repository storing the majority of official Haskell packages is called Hackage (`http://hackage.haskell.org/`). Exercise 2 gives some more details about using Cabal and sandboxes.

To use an already installed library in your Haskell program, use the import statement, for example:

```haskell
import Data.Map
```

To avoid name clashes, you can use the *qualified* import statement:

```haskell
import qualified Data.Map as Map
```

After including this statement into your program, you will be able to refer to the definitions from the `Data.Map` module using the syntax `Map.<function>` (e.g. `Map.fromList`).

Exercise 1: Cycles in Graphs

Write a program that verifies whether a *directed* graph contains any cycles. For the input file, feel free to pick whichever format fits your needs best (adjacency matrix, picture, etc.). You may read the graph out of a file or from standard input, and you may output the result into a file or onto the standard output.

Exercise 2: Parsing of Network Masks

Extend the port scanner from the last class with the possibility of specifying network masks. Thus, the user should additionally be able to write the following command:

```
scanner 1 10 'www.google.com[0-50]' '127.0.0.0/24[10-60]'
```

meaning: scan the ports 0 through 50 on the host `www.google.com` and the ports 10 through 60 on all

[*] All links are clickable.

machines in the subnetwork given by the mask `127.0.0.0/24`, using 10 threads and waiting 1 second before considering that the connection timed out.

You may use the following workflow scheme to build and test your code.[1]

1. Download `scanner.hs` and `scanner.cabal` from `http://lacl.fr/~sivanov/doku.php?id=en:haskell_for_life` and put them into the same directory. `scanner.hs` contains the source code, while `scanner.cabal` contains some metadata about the project, including the list of dependencies.
2. `cabal update` — fetch the latest list of available packages from Hackage.
3. `cabal sandbox init` — initialise a Cabal sandbox in the current directory in order to be able to install new libraries locally without altering your Haskell user profile.
4. `cabal install --only-dependencies` — install all the dependencies of the project described by the `.cabal` file from the current working directory. The project itself will neither be built, nor installed. If run in a directory containing a Cabal sandbox, this command will only install the libraries in the sandbox, without altering your Haskell user profile.
5. `cabal build` — build the project described by the `.cabal` file from the current directory. The resulting binary will be located in `dist/build/scanner/`.
6. `dist/build/scanner/scanner 1 10 'localhost[0-80]'` — run the port scanner.
7. `cabal repl` — start an interactive Haskell session (GHCi) and load the source of the project as well as all the necessary dependencies. Simply running `ghci` won't work, because in this case GHCi will look for libraries in your Haskell user profile, and not in the sandbox.

Should you need to add any new library to your project, feel free to append the name of the library and its version to the `build-depends` field in `scanner.cabal`. Don't forget to re-run `cabal install --only-dependencies` to actually add the library to your sandbox.

Exercise 3: Simple Arithmetic

Write a program which will read a list of expressions in reverse Polish notation from a file and which will output the result of each expression. For example, for the input

```
3 1 2-*
7 2 4/*
3 2 1+/
```

your program should output

```
-3.0
3.5
1.0
```

You may read the expressions from a file or from standard input, and you may output the result into a file or onto the standard output. Feel free to modify the format of the input and of the output as you find fit.

Bonus Modify your program to work with expressions written in normal infix notation. For example, the input to the modified version may look as follows:

```
(1 - 2) * 3
2/4*7
(1    + 2)/ 3
```

This input should yield the same output, that is:

```
-3.0
3.5
1.0
```

[1]Tested on a Linux system, but should work similarly on other platforms.