Informatics 1 — Functional Programming
Programming Exam

Instructions

1. The exam lasts two hours.
2. Place your student identity card face-up on the desk in front of you. The invigilator may come to check your identity, and in this case you may be asked to allow the invigilator to briefly use your computer. The exam time has been calculated to allow time for such interruptions.
3. You may log into your computer as soon as you are ready to do so.
4. To download the exam paper and template file, type the following in a terminal window:

   ```
   getpapers
   ```

   This will create subdirectories `inf1-fp-pe1/papers` and `inf1-fp-pe1/templates` in your home directory.
5. It is recommended that you take a copy of the `exam.hs` template file into your home directory:

   ```
   cd
   cp inf1-fp-pe1/templates/exam.hs $HOME
   ```

   This leaves an original copy of the `exam.hs` template in `inf1-fp-pe1/templates` should you need it.
6. A bug in DICE causes the keyboard to erroneously be set to US-style rather than UK-style on some machines. This causes some of the keys (for instance tilde, `~`) to produce a different character. Please look at the top right corner of your screen next to the day and time. You should see `en1` there. If you see `en2`, click on it to switch.
7. If you want to use Atom, type the following in a terminal window:

   ```
   link-atom-haskell
   ```

   Then start Atom in the normal way.

   Do nothing further until the start of the exam is announced!
8. When the start of the exam is announced, open the exam paper (it will prompt you to accept the `acroread` licence agreement):

   ```
   acroread inf1-fp-pe1/papers/exam.pdf
   ```

   Important: the directory `inf1-fp-pe1/papers` is read-only, so you cannot save your work in this directory.
9. Edit the file `exam.hs` to include your answers. To aid marking, your answers should appear in the same order as questions on the exam.

   Please Turn Over

10. You may use any tools available under DICE. You are recommended to save your work on a regular basis. Compile, run and debug your program as normal.

11. **Before submitting, make sure your file compiles correctly. Code which prevents the file from compiling should be made into comments. But please make sure that your file includes a definition of all of the required functions, at least along the lines of the starting definition (for instance, `f = undefined`) in the template file `exam.hs`.**

12. *Please ensure that the version of* `exam.hs` *you are about to submit definitely contains your solutions to the exam, for example using the* `head` *command to list the first lines of its contents:*

    ```
    head -25 exam.hs
    ```

    Submit your file using the command:

    ```
    examsubmit exam.hs
    ```

    If you get an error, please check carefully that your file is called `exam.hs` and that you are in the same directory as this file. If you continue to have problems, please contact one of the invigilators.

    Repeated submit commands are allowed, and will overwrite previous submissions. The last file submitted will be the one marked.

13. When the invigilators announce the end of the exam, you must submit and log out immediately.

Guidelines

- In the answer to any part of any question, you may use any function specified in an earlier part of that question. You may do this whether or not you actually provided a definition for the earlier part; nor will you be penalized in a later part if your answer to an earlier part is incorrect.

- You can write as many helper functions as you like, as long as they respect the restrictions of the question you are answering. For example, if a question states you can use list comprehension but not recursion, then you cannot use a recursive helper function, but you can use list-comprehensive helper functions. If a question states you cannot use library functions, you can still write helper functions that behave like library functions.

- Some part of your mark depends on whether your file compiles correctly (contains no syntax or type errors), and some depends on whether your functions return correct answers. You are encouraged to test your code, by checking that your functions behave according to the examples given, or using QuickCheck, and to include test code in your submitted solutions. Any text that is not legal Haskell, including partial answers that do not compile, should be included as comments. Correct solutions will receive full credit. Partial credit may be awarded for solutions that are partially correct, and for indicating how you have tested your work.

- As an aid to memory, *basic functions* are listed in Figure 1 and *library functions* are listed in Figure 2. You will not need all the functions listed. Haskell documentation is also available by pointing your browser at:

  `file:///home/s*******/inf1-fp-pe1/papers/index-haskell.html`

  where `s*******` is your s-matric number. Here you can find library functions and the Haskell language definition.

  (Note that internet access has been disabled.)

```
div, mod :: Integral a => a -> a -> a
(+), (*), (-) :: Num a => a -> a -> a
(/) :: Fractional a => a -> a -> a
(^) :: (Num a, Integral b) => a -> b -> a
(<), (<=), (>), (>=) :: Ord a => a -> a -> Bool
(==), (/=) :: Eq a => a -> a -> Bool
(&&), (||) :: Bool -> Bool -> Bool
not :: Bool -> Bool
max, min :: Ord a => a -> a -> a
isAlpha, isLower, isUpper, isDigit :: Char -> Bool
toLower, toUpper :: Char -> Char
digitToInt :: Char -> Int
intToDigit :: Int -> Char
even, odd :: Integral a => a -> Bool
```

Figure 1: Basic functions

```
sum, product :: (Num a) => [a] -> a
sum [1.0,2.0,3.0] = 6.0
product [1,2,3,4] = 24

and, or :: [Bool] -> Bool
and [True,False,True] = False
or [True,False,True] = True

maximum, minimum :: (Ord a) => [a] -> a
maximum [3,1,4,2] = 4
minimum [3,1,4,2] = 1

reverse :: [a] -> [a]
reverse "goodbye" = "eybdoog"

concat :: [[a]] -> [a]
concat ["go","od","bye"] = "goodbye"

(++) :: [a] -> [a] -> [a]
"good" ++ "bye" = "goodbye"

(!!) :: [a] -> Int -> a
[9,7,5] !! 1 = 7

length :: [a] -> Int
length [9,7,5] = 3

head :: [a] -> a
head "goodbye" = 'g'

tail :: [a] -> [a]
tail "goodbye" = "oodbye"

init :: [a] -> [a]
init "goodbye" = "goodby"

last :: [a] -> a
last "goodbye" = 'e'

takeWhile :: (a->Bool) -> [a] -> [a]
takeWhile isLower "goodBye" = "good"

take :: Int -> [a] -> [a]
take 4 "goodbye" = "good"

dropWhile :: (a->Bool) -> [a] -> [a]
dropWhile isLower "goodBye" = "Bye"

drop :: Int -> [a] -> [a]
drop 4 "goodbye" = "bye"

elem :: (Eq a) => a -> [a] -> Bool
elem 'd' "goodbye" = True

replicate :: Int -> a -> [a]
replicate 5 '*' = "*****"

zip :: [a] -> [b] -> [(a,b)]
zip [1,2,3,4] [1,4,9] = [(1,1),(2,4),(3,9)]
```

Figure 2: Some library functions