ANUC 1100 – Introduction to Programming and Algorithms

Semester 1, 2018

Assignment 1

Due date: 05:00pm, 18 April, 2018

This assignment will be marked out of 15. It will count for 15% of the final grade. Below you will find the specifications.

Instructions:

- All functions you developed for this assignment must be included in one single .hs file and submitted to the assignment dropbox link on Wattle by 05:00pm 18 April, 2018. Make sure you include your name and student ID in the comment section at the top of your source file. If you have any other supporting documents, you can zip all files into a single archive Assignment_1_<name>_<ID>.zip where <name> and <ID> are your details.
- Late submission without an extension are penalised at the rate of 5% of the possible marks available per working day or part thereof. The assignment is not accepted after 10 working days after the due date.
- Plagiarism will attract academic penalties in accordance with the ANU guidelines.

Good luck and enjoy the time you will spend on this assignment

Overview

In this assignment, you will be implementing a series of small functions which test your programming skills in Haskell. So far in the course, you have learned typed expressions, conditional branching and recursion. We have also covered basic data abstract types and user defined types. You will use the acquired skills to solve two little programming problems. Your code should follow good coding standards with meaningful variable and functions name. It must also be compiled without any error or warning.

For the problems in which you are required to write some functions:

- Define each function with the exact name and type specified. You can (and in most cases you should) define each function using a number of simpler functions
- For each function definition, provide a type profile and comprehensive comments where appropriate.
- Unless said otherwise, functions must be defined for all of its input values.
- Use the error function or undefined in which a function should terminate with an error message

Problem 1 (8 marks) – Validating Credit Card Numbers

Most of people nowadays actually own or at least are familiar with bank cards. Credit card is one of the most popular types of bank cards which allow the holder to purchase goods or services on credit. In the context of electronic commerce, customers can conveniently pay for their shopping using their credit card. They first enter their credit card number, follows by the account holder's name, card expiry date and a security code. The online payment application will first check whether the card number alone is valid or not to protect against accidental errors. If it is, the issuing bank will be contacted to validate the other details of the card as well as the card balance.

A valid credit card number has a length from 13 to 16 digits which are grouped into three segments: Bank Identification Number (BIN), Account Number and Check Digits. MOD 10 is the algorithm commonly used to validate credit card numbers.

Suppose that we want to check if the number 38520000023237 could be a real credit card number. The validation process has the following steps:

- Take the last digit: 7. This is called the check digit
- Take the rest of the sequence: 3852000002323
- Double every other digit starting from the right: 6,8,10,2,0,0,0,0,0,2,6,2,6
- Sum the digits of the above numbers (For example, with number 10, we have two digits: 1 and 0. The sum of the digits in number 10 is 1 + 0 = 1): 6,8,1,2,0,0,0,0,0,2,6,2,6
- Add all the above digits together: 6+8+1+2+0+0+0+0+0+2+6+2+6 = 33
- Multiply the result by 9: 33 * 9 = 297
- Take the last digit of the result: 7. If this matches the check digit, you have a valid credit card sequence.

Since our heck digit 7 matches our result 7, you can conclude that the given sequence is a valid credit card number.

In this problem, you will write a Haskell program that implements the MOD 10 algorithm and validates a given credit card number. You can take advantage of pre-defined functions from the Data.Char and Data.List modules in Prelude, by putting the following lines to the top of your source code:

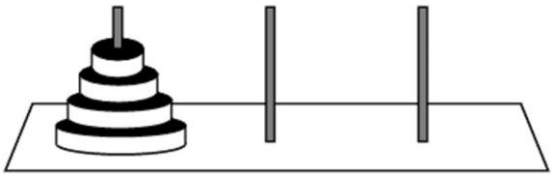
```
import Data.Char
import Data.List
```

You should implement the following functions in your program:

Function	Mark
text2digits – This function converts a text into a list of integers. The output type should be Maybe [Int] so that the function will return Nothing if the input text has any non-digit character.	3
is_valid_card – this function accepts a sequence of digits (in form of a string) and returns a Boolean value indicating whether the given sequence could be a valid credit card number.	5

Problem 2 (2 marks) – Solving the Tower of Hanoi Game

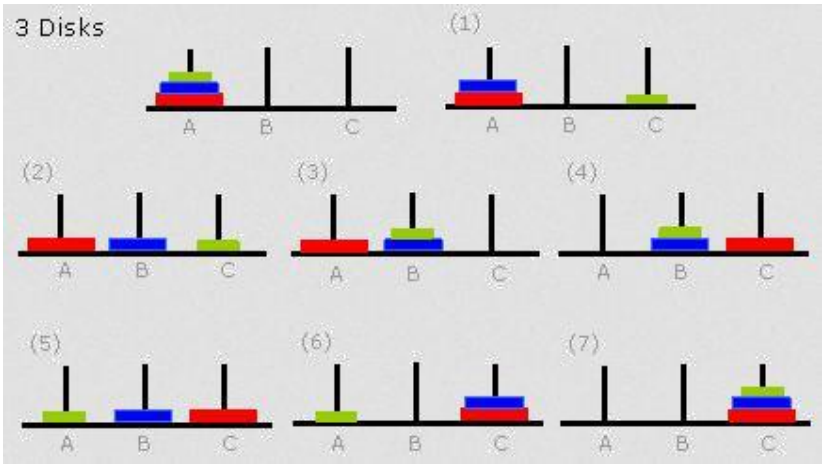
The Tower of Hanoi (sometimes called Lucas's Tower, is a puzzle game which has three rods and a number of disks of different sizes. The game starts with the disks stacked on one rod in the ascending order of size, thus making a conical shape:



A player needs to move the entire stack to another rod, following the below rules:

- Only one disk can be moved at a time
- Each move consists of taking the top disk from one stack and placing it on top of another stack
- No disk may be placed on top of a smaller disk.

A simple three-disk game can be solved in 7 steps as follows:



For this problem, you are provided with some source code which has basic setups for the game:

- Basic data structures: Game, Stack, Disk
  - Function is_valid_game that checks if a game state is valid (e.g. for any stack, no disk is placed on top of the smaller disk)
  - Function print_game that prints the game states, converting the initial state to the winning state

An acceptable solution to this game will print the following output on the screen:


```
...
1
2
3
4
5
stack_1  stack_2  stack_3

                  1
                  2
                  3
                  4
                  5
stack_1  stack_2  stack_3
```

Your task in this problem is to research how this game can be solved and implement the solution in Haskell based on the given source code. You can find several methods other people used for this game on the Internet. If you actually build your solution based on other people's ideas, make sure you acknowledge that in your submission.

Coding standards and Referencing

Your code should follow the coding standards below:

- Space-indented source files: make sure you configure your editor correctly so that it saves your files with hard-spaces rather than tabs. If you do not remember how this can be done, review the first lab.
- No piece-wise function definitions: while it is actually common in the Haskell community to write functions by providing multiple versions of the function definition, which all cover only some parts of the input space, it is discouraged here for quite a number of reasons which are detailed in your lecture slides. The rule which applied here is: the patterns which you use on the left side of a function definition have to match the complete input space. This is always true if you use just identifiers for your parameters.
- Clear names for everything: A clear name for a function combined with clear names inside the pattern which matched the inputs documents most functions well and often sufficiently.
- Usage of the Haskell Prelude (or other standard Haskell packages) is ok (without higher order functions such as map, fold, filter, ect).
- No predefined, higher order functions: if you want to use higher order functions, then you must implement them yourself. This restriction is necessary to allow us to see whether you know what you are doing.
- Comments where necessary: Never comment the obvious or a line-by-line basis. Instead, comment in larger blocks where the functionality is no longer immediately obvious.

Please note that any form of code copying and exchange is not allowed and the code they submit should be your own work. In addition, never directly exchange source code with other students. If your code is found in any somebody' else works, you will become part of a plagiarism case.

Deliverables and Program Demonstration

All functions you developed for this assignment must be included in one single .hs file and submitted to the assignment dropbox link on Wattle. Make sure you include your name and student ID in the comment section at the top of your source file. If you have any other supporting documents, you can zip all files into a single archive Assignment_1_<name>_<ID>.zip where <name> and <ID> are your details.

During the lab time of the week of submission, you will demonstrate the implementation of your code. The idea is to allow you to demonstrate your understanding of your program. Some questions you may want to address during the demonstration are:

- What is your rationale for your program designs? Did you consider alternatives? Which? What makes your solutions efficient and/or elegant? Or: are you solutions less efficient but more readable?
- How did you come up with your solutions?
- Did you learn anything during the assignment?

The code demonstration session will be ten minutes for each student.

Marking

All of the total mark is allocated to the code submitted, awarded to code functionality (i.e. it works), clarity of the code (i.e. we immediately understand why it works), and efficiency (i.e. it returns a solution within a reasonable time). The actual weighting for each question as follows:

Problem 1: 8 marks
Problem 2: 2 marks

Though there is no mark for the code demonstration, it is critical that you can demonstrate your understanding of your submitted program. This can indicate whether the submitted code is actually your work or not.