Functional Programming and Verification

Prof. Dr. H. Seidl, N. Hartmann, R. Vogler
WS 2018/19

Exercise Sheet 9
Deadline: 06.01.2019

General Information

Detailed information about the lecture, tutorials and homework assignments can be found on the lecture website[1]. Solutions have to be submitted to Moodle[2]. Upload your solutions as a single file named 'hw09.ml' to moodle. Since submissions are tested automatically, solutions that do not compile or do not terminate within a given time frame cannot be graded and thus result in 0 points. If you do not manage to get all of your implementations compiling and/or terminating, comment them out and use the original definition (if you comment out the definition, the tests won't compile)! Use Piazza[3] to ask questions and discuss with your fellow students.

Functional Programming

Since this is a course about functional programming, we restrict ourselves to the functional features of OCaml. In other words: The imperative and object oriented subset of OCaml must not be used.

Assignment 9.3 (H) Santa's Christmas Present Factory [20 Points + 10 Bonus]

In preparation for the upcoming christmas holidays, Santa needs your help to automate his christmas present factory, to make sure the children all around the world get their wishes fulfilled.

Hint: Check the end of this assignment for some hints Santa has for you.

1. During the year, Santa keeps track of all children's behavior in his notebook. So your first task is to read his notes into your program such that this information is available. The structure of his notebook is such that every line contains a child's name and its behavior (either *nice* or *naughty*) separated by a colon. Implement a function `read_notes : string -> notes` which reads the file with the given name. The `notes` type has the following definition:

   ```
   type behavior = Nice | Naughty
   type notes = (string * behavior) list
   ```

   If this function encounters an invalid entry while reading the file, the exception

   ```
   exception Invalid_file_format of string
   ```

   has to be thrown with the invalid file's name as the argument. We consider an entry invalid if it does not stick to the form *name:behavior* where *name* is non-empty and *behavior* is either *nice* or *naughty*. [3 points]

[1] https://www.in.tum.de/i02/lehre/wintersemester-1819/vorlesungen/functional-programming-and-verification/
[2] https://www.moodle.tum.de/course/view.php?id=44932
[3] https://piazza.com/tum.de/fall2018/in0003/home

2. The most time-consuming part of Santa's working day is reading all the wish lists. Luckily, over the last years, Santa trained all children to write their wish lists in such a way that every line has the form *wish:importance*, where *wish* is simply the name of one toy and *importance* is a value between 1 and 100 that gives Santa a hint on how desperately the child wants to have this present. Write a function `read_wishlist : string -> (string * int) list` that reads the wish list with the given file name. Again throw an `Invalid_file_format` exception if any entry is not valid. [3 points]

3. Like every year, Santa has asked his many helpers to write down all toys that are available this year in a large catalogue. There is no harm in loading this catalogue into your program as well. Every entry in the catalogue file is of the form *toy:weight*[4] where *weight* is a positive integer weight of the *toy* that Santa needs to know to prevent the overload of his sleigh. Implement a function

   ```
   load_catalogue : string -> (string * int) list
   ```

   that reads the given file and again throws an `Invalid_file_format` exception if there is something wrong with the file. [3 points]

4. The next thing to do is to give Santa a means to print a list of presents he has chosen for a child, so that his helpers can load everything into the sleigh. Implement the function `write_list : string -> string list -> unit` that stores the list of selected presents (2nd argument) into a file (1st argument). [2 points]

5. Lastly, all naughty kids do not receive any presents, instead Santa sends them a letter that they have to be more polite and stick to their programming exercises to receive any presents next year. Santa does not like this part of his job, so he really wants this to be automated. Implement the function `write_letter : string -> unit` that writes some harsh words[5] into the file with the given name. [1 point]

6. Now everything is together to run Santas christmas present factory. Implement the function `run_santas_factory : int -> selection_alg -> unit` that performs the following tasks:

   (a) Load the toy catalogue from the file `"toys_catalogue.txt"`.

   (b) Read Santa's notes from the file `"santas_notes.txt"`.

   (c) For every naughty child *name* write a letter to the file `"name_letter.txt"`.

   (d) For every nice child *name*:

      i. Read the child's wish list from the file `"name_wishlist.txt"`.

      ii. Construct a `(string * int * int) list` which has an entry `(toy,importance,weight)` for all the child's wishes, where the first two parts are taken from the wish list and the `weight` is checked in the catalogue. Toys on the child's wish list that are not available (in the catalogue) have to be removed from the list.

      iii. Pass this list to the present selection algorithm, together with the sleigh's capacity (passed as the 2nd resp. 1st argument to `run_santas_factory`).

      ```
      type selection_alg = (string * int * int) list
                -> int -> string list
      ```

[4] Santa really is into this format, huh?
[5] Write anything, be creative!

iv. Write the selected presents to the file `"name_presents.txt"` one per line.

Errors are handled in the following way: If a child's wish list is invalid, that child is simply ignored and everyone else still has to get their presents! All other exceptions are not handled inside `run_santas_factory`. [8 points]

7. Implement the function

```
knapsack : (string * int * int) list -> int -> string list
```

as a possible present selection algorithm. The function has to find an optimal selection of presents from the list (1st argument) such that the total weight is less than or equal to the capacity of the sleigh (2nd argument) and that the sum of *importance* of all selected presents is maximal. [10 points]

Hint: First, read all tasks of this assignment carefully, as you may identify common functionality that may be reused in different places.

Hint: Before you start to implement your solution, check the provided input and output text files in the folder.

Hint: Assume that everything is in lower case letters. No need to care about case-insensitive comparison.

Hint: Make sure to always close all opened files (even in case of exceptions)!

Hint: Check the OCaml ***`String`*** *documentation for the function* ***`split_on_char`****.*

Hint: Use the function ***`int_of_string`*** *to convert a string to an integer. Be aware that this function may throw. Check the documentation.*