Problem Set 0: Getting Started with OCaml

Points: 6

Getting Started

First, you need to install OCaml and some standard tools for working with OCaml projects, including: - opam, an OCaml package manager - dune, an OCaml build tool

To do you can follow either of these guides: - https://cs3110.github.io/textbook/chapters/preface/install.html - https://ocaml.org/docs/up-and-running

Note: if you follow the first guide, you may still need to install `dune` afterward, which is straightforward. Open a terminal and run the command `opam install dune`.

Once you're done with that, you're ready to get started on the assignment.

These are a few warmup problems for getting up to speed writing functions in OCaml. Write your code in the file **`src/bin/main.ml`**. To test your code compile and run it with

```
> cd src/
> dune exec bin/main.exe test
```

These will run the tests at the bottom of the `main.ml`. You will submit your work by uploading the `main.ml` file to GradeScope.

NOTE: you are strongly encouraged to take a look at the OCaml standard library manual pages and use any functions that might be helpful. For example, when writing functions that manipulate strings or characters, you might want to look at the Char library and String library. The standard library is actually pretty short (for better or worse), so you can actually take a look and learn a lot by reading about the functions.

If you find the English description of the functions you are asked to write to be confusing, feel free to look at the tests for additional examples. For instance, the `isDigit` problem has a test case of the form:

```
let isDigitTest1 () = isDigit ’0’
```

Each test should return `true`. So this is telling us that `isDigit '0'` should return `true`. While the following test for problem 2:

```
let newFileNameTest1 () = newFileName "A.java" "class" = "A.class"
```

is testing whether when we apply your `newFileName` function to these arguments, it in fact gives back the result on the right.

Problems

1. (.25 Points) Write a function `isDigit : char -> bool` such that a call `(isDigit c)` returns `true` if `c` is a digit character. If `c` isn't a digit character`isDigit` should return `false`.

2. (.25 Points) Write a function `newFileName : string -> string -> string` such that a call `(newFileName oldFileName newExtension)` returns `oldFileName` but with the new extension. For example, the call `(newFileName "MyFile.java" "class")` should return the string `"MyFileName.class"`. The input file name may have zero or more dots.

   - The call `(newFileName "MyFile" "anything")` (i.e., with no dots) should return just `"MyFile"`;
   - The call `(newFileName "My.File.java" "class")` should return `"My.File.class"` (i.e., only the rightmost dot matters);
   - The call `(newFileName "MyFile." "class")` should return `"MyFile.class"`, i.e., a trailing dot with no actual file extension still gives the new extension.

   Hint: Take a look at `String.rindex_opt` in the standard library.

3. (.25 Points) Write a function `homePath : unit -> string list` that returns a list of the names of the directories in the path to your home directory. For example, on my computer the call `(home ())` would return the list `["home";  "joe"]`. This is an exercise in looking things up. You might want to google "unix home directory variable" and take a look at OCaml's `Unix` and `String` modules.

4. (.25 Points) Write a function `atoi : string -> int` (i.e., "ASCII to integer") such that a call of the function such as `(atoi "59")` would return the integer `59`. You may assume that the characters in the string are all decimal digits. **Hint:** Consider using the function `Lib.explode : string -> char list`. For example, `(Lib.explode "314")` returns the list of characters `['3'; '1'; '4']`.

5. (1 Points) Write a function `tokenize : string -> token list` such that a call `(tokenize string)` returns a list of tokens, where the type of tokens is defined as:

   ```
   type token = And | Or | If
   ```

   Your function should convert the string `"&&"` to the token `And`; the string `"||"` to the token `Or`; and the string `"if"` to the token `If`. For example

   ```
   (tokenize "  || if       &&")
   ```

   should return the list `[Or; If; And]`. Note that white space in the form of the space character `_` should be ignored. You can assume that the only space characters in your input are the ones that occur in `&&`, `if`, and `||`, and you don't have to handle incomplete or partial tokens. For example, your code is allowed to return any list you like on the input `&|`. **Hint**: Use the `Lib.explode` function.

Problem 6 relates to binary trees with interior nodes `Arrow` and with leaf nodes that are either the constant `C` or a variable `Var 0`, `Var 1` etc.

```
type t = C
       | Var of int
       | Arrow of { from : t
                  ; too  : t
                  }
```

Note that the 'too' field should probably be called 'to' (that is the names would be from and to), but 'to' is already a reserved keyword in OCaml, so we cannot use it for a field name.

With this definition, we can express trees as follows.

```
# C;;
- : t = C

# Var (2 + 3);;
- : t = Var 5

# Arrow { from = C; too = C};;
- : t = Arrow {from = C; too = C}
```

We'll use the following examples.

```
let t0 = Arrow { from = Var 0; too = Arrow { from = C; too = C }}
let t1 = Arrow { from = C; too = Var 1 }
let t2 = Arrow { from = t1; too = Arrow { from = C; too = C }}
let t3 = Arrow { from = C; too = t0 }
```

```
         t0 = ->          t1 = ->            t2 = ->                 t3 = ->
             /  \               /  \               /  \                    /  \
           v0    ->            C    v1           t1    ->                C    t0
                /  \                                   / \
               C    C                                 C   C
```

6. (1 Point) Write the function `formatTree : t -> string` which returns a string representation of a tree. For example, with `t0` thru `t3` as above, we have

```
# (formatTree (Var 8));;
- : string = "v8"

# (formatTree t0);;
- : string = "(v0 -> (C -> C))"
```

```
# (formatTree t1);;
- : string = "(C -> v1)"

# (formatTree t2);;
- : string = "((C -> v1) -> (C -> C))"

# (formatTree t3);;
- : string = "(C -> (v0 -> (C -> C)))"
```

7. (.75 Point) A palindrome is a string that is the same when read forwards or backwards, such as “racecar” or “level”. Write a function `subpalindrome : string -> string` such that a call `(subpalindrome s)` returns the longest palindrome that can be obtained from `s` by removing an equal number of characters from the beginning and end of `s`. For example, if `s` is `wracecare`, `(subpalindrome s)` should return `racecar`, which is obtained from `s` by deleting 1 character from the beginning and end. If `s` is already a palindrome, `(subpalindrome s)` should just return `s`.

8. (.5 Points) The type `comparison` is defined as:

   ```
   type comparison = GEQ | LT
   ```

   Where `GEQ` represents “greater than or equal” and `LT` represents “less than”. Write a function `listComparisons : int list -> comparison list` such that a call `(listComparisons [n_1; n_2; n_3; ...; n_k])` returns a list `[c_1; c_2; ...; c_k]`, where `c_1` is always `GEQ`, and for each `i`, `c_(i+1)` is `GEQ` if `n_i <= n_(i+1)`, and `LT` otherwise.

9. (1 points) Write a function `patience : (int list) list -> int -> (int list) list`. When running `(patience ls n)`, the first argument `ls` is a list of lists of integers. We call each list in `ls` a pile. You may assume that each pile is non-empty, and that each pile is sorted in ascending order. Finally, you may also assume that if `ls` is of the form `[p_1; p_2; ...; p_n]`, then the head of `p_i` is strictly less than the head of `p_(i+1)`.

   Evaluating `(patience ls n)` should return a list of piles obtained from taking `ls` and inserting `n` so that either `n` has (1) been added to the head of the first pile in `ls` whose previous head is greater than or equal to `n`, or (2) if no such pile exists, then a new pile containing just `n` is added to the end of `ls`.

   For example, `patience [[4]; [5]] 3 = [[3;4]; [5]]`, and `patience [[2]; [6]] 4 = [[2]; [4;6]]`, and `patience [[3]] 4 = [[3]; [4]]`

10. (.75 points) In this problem, you will implement the `FilteredSet` module in `main.ml`. A filtered set consists of a filter function `f : int -> bool` and a set of integers, all of whose elements return `true` when passed to `f`. Evaluating `newSet f` returns an empty set with filter function `f`. If `s` is some filtered set with filter `f` then `insert n s` returns a filtered set with the same filter function `f`, and all of the elements of `s`, and additionally adds

`n` if `f n = true`, and otherwise omits `n`. Evaluating `member n s` returns `true` if `n` is in the set `s` and `false` otherwise. Finally `mapAndFilter g s` returns the set obtained by applying `g` to all of the elements of `s` and removing the ones for which the filter function no longer will return true.