SDE 2: Functional Programming - OCAML

1. Preface

The overall objective of SDE2 is to implement 6 sets of functions corresponding to 6 list-based problems. Three of these problems were inspired by actual coding tests given during interviews for positions with companies such as Google, Amazon, Microsoft, and Apple. There are YouTube videos that describe three of these problems listed here. Watching related videos on YouTube is recommended to better understand the problem requirements. This project is to be done using a purely functional programming paradigm and OCaml. Note that all problems must be solved using OCaml lists, even if the original problem references arrays or strings. The problems are:

1. (16pts) Write a function `first_duplicate`.
2. (16pts) Write a function `first_nonrepeating`.
3. (16pts) Write a function `sumOfTwo`.
4. (16pts) Write a function `take`.
5. (16pts) Write a function `drop`.
6. (20pts) Write a function `powerset`.

This should actually be some fun and really good experience with coding interview tests. Each of the problems is described separately, with examples.

2. The Problems and Required `OCaml` Functions

Pay special attention to the naming and argument(s) of each required OCaml function.

2.1 First Duplicate in a List (`first_duplicate`)

Find and return the first duplicate in an integer list. You can see the problem formulation at: https://www.youtube.com/watch?v=XSdr_O-XVRQ

Prototype:
`first_duplicate` of a list returns `-10000` if there are no duplications in the integer list argument. Otherwise, the first item that is the same (duplicate) of another item already passed in the list is returned. Note this is NOT the first item with a duplicate farther down the list; rather, it is the first item with a duplicate BEFORE it on the list.

Signature:
```
val first_duplicate : int list -> int = <fun>
```

Sample Use:

```
# first_duplicate [1;2;3;4;5;6;7;4;5;8;9];;
: int = 4
# first_duplicate [1;2;3;4;5;6;7;8;5;2;9];;
: int = 2
# first_duplicate [1;2;3;4;5;6;7;8;9;10];;
: int = -10000
```

2.2 First Non-Repeating Element in a List (`first_nonrepeating`)

Find the first item that is not in the integer list more than once. You can see the problem formulation at: https://www.youtube.com/watch?v=5co5Gvp_-S0

Prototype:
`first_nonrepeating` of a list returns `-10000` if there are no non-repeated (non-duplicated) elements in the list. Otherwise, it returns the first non-repeating element in the integer list.

Signature:
`val first_nonrepeating : int list -> int = <fun>`

Sample Use:

```
# first_nonrepeating [1;2;3;2;7;5;6;1;3];;
: int = 7
# first_nonrepeating [1;2;9;3;2;7;5;6;1;3];;
: int = 9
# first_nonrepeating [1;2;9;3;2;7;5;6;10;30];;
: int = 1
# first_nonrepeating [1;2;9;3;2;7;5;6;1;10;30];;
: int = 9
# first_nonrepeating [1;2;9;3;2;7;5;9;6;1;10;30];;
: int = 3
# first_nonrepeating [1;2;3;2;7;5;6;1;3];;
: int = 7
# first_nonrepeating [1;2;3;4;5;1;2;3;4;5];;
: int = -10000
# first_nonrepeating [1;2;3;4;5;1;2;3;4;9];;
: int = 5
# first_nonrepeating [1;1;1;2;2;2];;
: int = -10000
```

2.3 The Sum of 2 Problem (`sumOfTwo`)

Look for a pair of elements from two integer lists that sum to a given value. You can see the problem formulation at:
https://www.youtube.com/watch?v=BoHO04xVeU0
https://www.youtube.com/watch?v=sfuZzBLPcx4

Two arrays contain numbers able to produce a given sum. For example, if `a = [1;2;3]`, `b = [10;20;30;40]`, and `v = 42`, then `sumOfTwo(a, b, v) = true` because `(2 + 40) = 42 = v`.

Prototype:
`sumOfTwo(a, b, v)` returns `false` if there does not exist an integer in `a` which, added to any integer in `b`, equals `v`. If there is such a pair, it returns `true`.

Signature:
```
val sumOfTwo : int list * int list * int -> bool = <fun>
```

Sample Use:

```
# sumOfTwo([1;2;3],[10;20;30;40],42);;
: bool = true
# sumOfTwo([1;2;3],[10;20;30;40],40);;
: bool = false
# sumOfTwo([1;2;3],[10;20;30;40],41);;
: bool = true
# sumOfTwo([1;2;3],[10;20;30;40],43);;
: bool = true
# sumOfTwo([1;2;3],[10;20;30;40],44);;
: bool = false
# sumOfTwo([1;2;3],[10;20;30;40],11);;
: bool = true
# sumOfTwo([1;2;3],[10;20;30;40],15);;
: bool = false
```

2.4 `take` Function

Signature:
```
val take : int * 'a list -> 'a list = <fun>
```

Prototype:
The function `take n lst` returns the first `n` elements of `lst`. If `lst` has fewer than `n` elements, return all of them. If `n < 0`, return an empty list.

Sample Use:

```
# take (2, [1;2;3;4]);;
: 'a list = [1;2]

# take (15, [1;2;3;4]);;
: 'a list = [1;2;3;4]

# take (-1, [1;2;3;4]);;
: 'a list = []
```

2.5 `drop` Function

Signature:
`val drop : int * 'a list -> 'a list = <fun>`

Prototype:
The function `drop n lst` returns all but the first `n` elements of `lst`. If `lst` has fewer than `n` elements, return an empty list. If `n < 0`, return an empty list.

Sample Use:

```
# drop (2, [1;2;3;4]);;
: 'a list = [3;4]

# drop (2, [1;2;3;4;5;6]);;
: 'a list = [3;4;5;6]

# drop (15, [1;2;3;4]);;
: 'a list = []

# drop (-1, [1;2;3;4]);;
: 'a list = []

# drop (0, [1;2;3;4]);;
: 'a list = [1;2;3;4]
```

2.6 `powerset` Function

Signature:
`val powerset : int list -> int list list = <fun>`

Prototype:
The `powerset` function takes a set `S` represented as a list and returns the set of all subsets of `S`. The order of subsets in the powerset and the order of elements in the subsets do not matter.

Hint: Consider the recursive structure of this problem. Suppose you already have `p`, such that `p = powerset s`. How could you use `p` to compute `powerset (x::s)`?

Sample Use:

```
# powerset [1;2;3];;
: int list list = [[]; [1]; [2]; [3]; [1;2]; [1;3]; [2;3]; [1;2;3]]

# powerset [1;2];;
: int list list = [[]; [1]; [2]; [1;2]]

# powerset [1;2;3;4];;
```

```
: int list list = [[]; [1]; [2]; [3]; [4]; [1;2]; [1;3]; [1;4]; [2;3];
[2;4]; [3;4]; [1;2;3]; [1;2;4]; [1;3;4]; [2;3;4]; [1;2;3;4]]
```

3. Resources

To successfully complete this SDE, it's highly recommended to take some time to review the following resources:

- Course slides on OCaml
- The OCaml website (https://ocaml.org)

4. OCaml Functions and Constructs Not Allowed

Of extreme significance is the restriction of the paradigm to pure functional programming (no side effects). **No ocaml imperative constructs are allowed.** Recursion must dominate the function design process. So that we may gain experience with functional programming, only the applicative (functional) features of ocaml are to be used. Please reread the previous sentence. This rules out the use of ocaml's imperative features.

To force you into a purely applicative style, **`let` should be used only for function definition. `let` cannot be used in a function body.** Loops and local or global variables are prohibited. The allowable functions in SDE1 are only those non-imperative functions in the Pervasives module and these individual functions listed below:

- List.hd
- List.tl
- List.cons
- List.nth
- List.mem
- List.append

This means you may not use the Array Module. Finally, the use of sequence-based control structure (Section 7.2 in OCaml manul: https://ocaml.org/manual/5.2/expr.html#ss%3Aexpr-control) is not allowed. Do not design your functions using sequential expressions. Here is an example of a sequence in a function body:

```
let print_assignment = function(student,course,section) ->
    print_string student; (* first you evaluate this*)
    print_string " is assigned to "; (* then this *)
    print_string course; (* then this *)
    print_string " section " ; (* then this *)
    print_int section; (* then this *)
    print_string "\n;; (* then this and return unit*)
```

If you have any questions or uncertainties, don't hesitate to reach out - I'm happy to provide clarification or guidance. The main goal of this assignment is to build

your skills in functional programming. Instead of relying on built-in OCaml functions or shortcuts that might oversimplify the task, focus on embracing the problem-solving process and fully exploring functional programming concepts.

5. Evaluation Criteria

Your solution will be tested against sample inputs. Each function should operate correctly with no errors or warnings. Grades will be based on functionality, adherence to specifications, and appropriate use of functional programming principles.

The grade is based upon a correctly working solution. We will also look for offending let use and sequences.

6. Submission Format

The final zipped archive is to be `<yourname>_sde2_F24.zip`, where `<yourname>` is your (CU) assigned user name. You will upload this archive to the Canvas prior to the deadline. The minimal contents of this archive are as follows:

1. **README.txt**: A file listing the contents of the archive and a brief description of each file. Include **the pledge** below:

Pledge: On my honor, I have neither given nor received aid on this SDE, including assistance from AI tools.

2. **OCaml Source Code**: A single file named `sde2.ml` with all required and any additional functions implemented. Note this file must include all the required functions, as well as any additional functions you design and implement. We will supply the testing data.

This means, among other things, that the code you submit is yourcode.

3. **Log File**: `log.pdf`, including the screenshot of sample uses of each required function. You can include all screenshots in a word file and convert it to the PDF format.

Final Remark: Deadline Matters

This reminder is included in the course syllabus. Since Canvas allows multiple submissions, you're encouraged to upload your progress as a standalone archive before the deadline, even if you haven't completed all the functions yet. This way, you can still qualify for partial credit while continuing to improve your solution.