AN IMPLEMENTATION OF LIST WITH AN EFFICIENT CONCATENATION

(56/56 points)

Concatenating two standard OCaml lists takes a time proportional to the length of the first list. In this exercise, we implement a data structure for lists with a constant time concatenation.
The preludes gives a type `'a clist`, which is either a single element of type `'a`, the concatenation of two `'a clist` or an empty `'a clist`.
This representation of a list is not linear: it is a tree-like datastructure since the `CApp` constructor contains two values of type `'a clist`.
The sequence of elements contained in a value of type `'a clist` is obtained by a depth-first traversal of the tree. For instance, the example given in the prelude, of type `int clist` is a valid representation for the sequence `[1;2;3;4]`.

1. Write a function `to_list : 'a clist -> 'a list` which computes the `'a list` that contains the same elements as the input `'a clist`, in the same order.
2. Write a function `of_list : 'a list -> 'a clist` which computes the `'a clist` that contains the same elements as the input `'a list`, in the same order.
3. Write a function `append : 'a clist -> 'a clist -> 'a clist` such that:
   1. `append CEmpty l = append l CEmpty = l`
   2. `append l1 l2 = CApp (l1, l2)` otherwise
4. Write a function `hd : 'a clist -> 'a option` that returns `Some x` where `x` is the first element of the input `'a clist`, if it is not empty, and returns `None` otherwise.
5. Write a function `tl : 'a clist -> 'a clist option` that returns `Some l` where `l` is the input sequence without its first element, if this input sequence is not empty, or returns `None` otherwise.

THE GIVEN PRELUDE

```
type 'a clist =
  | CSingle of 'a
  | CApp of 'a clist * 'a clist
  | CEmpty

let example =
  CApp (CApp (CSingle 1,
              CSingle 2),
        CApp (CSingle 3,
              CApp (CSingle 4, CEmpty)))
```

YOUR OCAML ENVIRONMENT

```
  in
  transf l []
;;

let rec of_list l =
  let rec transf l liste = match liste with
    | [] -> CEmpty
    | [a] ->  CSingle a
    | a :: xs -> CApp (CSingle a, transf (CSingle a) xs)
  in
  transf CEmpty l
;;

let append l1 l2 = match l1, l2 with
  | CEmpty, l -> l
  | l, CEmpty -> l
  | l1, l2 -> CApp (l1, l2)
;;


let hd l =
  let liste = to_list l in
  match liste with
  | [] -> None
  | x :: xs -> Some x
;;

let tl l =
```


Exercise complete (click for details) 56 pts

Exercise 1: `to_list` Completed, 11 pts

Found `to_list` with compatible type.

Computing `to_list (CApp (CApp (CSingle 3, CApp (CEmpty, CSingle 2)), CEmpty))`

Correct value `[3; 2]` 1 pt

Computing `to_list (CSingle (-2))`

Correct value `[-2]` 1 pt

Computing
`to_list (CApp (CApp (CSingle (-5), CApp (CEmpty, CSingle (-3))), CSingle 0))`

Correct value `[-5; -3; 0]` 1 pt

Computing
```
  to_list
    (CApp (CApp (CEmpty, CApp (CSingle (-4), CSingle (-4))),
      CApp (CSingle 4, CSingle (-2))))
```

Correct value `[-4; -4; 4; -2]` 1 pt

Computing `to_list (CApp (CApp (CEmpty, CApp (CEmpty, CSingle 3)), CSingle (-3)))`

Correct value `[3; -3]` 1 pt

Found `to_list` with compatible type.

Computing `to_list CEmpty`

Correct value `[]` 1 pt

Computing `to_list (CApp (CSingle 'y', CEmpty))`

Correct value `['y']` 1 pt

Computing `to_list (CApp (CApp (CEmpty, CApp (CSingle 'w', CEmpty)), CEmpty))`

Correct value `['w']` 1 pt

Computing `to_list (CApp (CApp (CEmpty, CSingle 'g'), CEmpty))`

Correct value `['g']` 1 pt

Computing `to_list (CApp (CEmpty, CApp (CEmpty, CApp (CEmpty, CSingle 'x'))))`

Correct value `['x']` 1 pt

Computing `to_list (CSingle 'd')`

Correct value `['d']` 1 pt

Exercise 2: `of_list` Completed, 11 pts

Found `of_list` with compatible type.

Computing `of_list [-2; 0; -2; 4]`

Correct value `(CApp (CSingle (-2), CApp (CSingle 0, CApp (CSingle (-2), CSingle 4))))` 1 pt

Computing `of_list [2; -3]`

Correct value `(CApp (CSingle 2, CSingle (-3)))` 1 pt

Computing `of_list [0; -1; -2; -1; -5; -1]`

Correct value 1 pt
```
  (CApp (CSingle 0,
    CApp (CSingle (-1),
     CApp (CSingle (-2),
      CApp (CSingle (-1), CApp (CSingle (-5), CSingle (-1)))))))
```

Computing `of_list [3; -2; -1; -4; 0; 4; 4; -4]`

Correct value 1 pt
```
  (CApp (CSingle 3,
    CApp (CSingle (-2),
     CApp (CSingle (-1),
```

Correct value 1 pt

```
(CApp (CSingle (-4),
  CApp (CSingle 3,
   CApp (CSingle (-5),
    CApp (CSingle (-3),
     CApp (CSingle 0,
      CApp (CSingle 0,
       CApp (CSingle (-5), CApp (CSingle 2, CApp (CSingle (-2), CSingle 3))))))))))
```

Found `of_list` with compatible type.

Computing `of_list []`

Correct value `CEmpty` 1 pt

Computing `of_list ['r']`

Correct value `(CSingle 'r')` 1 pt

Computing `of_list ['n'; 'k']`

Correct value `(CApp (CSingle 'n', CSingle 'k'))` 1 pt

Computing `of_list ['v']`

Correct value `(CSingle 'v')` 1 pt

Computing `of_list ['s'; 's'; 'q'; 'p']`

Correct value `(CApp (CSingle 's', CApp (CSingle 's', CApp (CSingle 'q', CSingle 'p')))` 1 pt

Computing `of_list ['v'; 'p'; 'r'; 'p']`

Correct value `(CApp (CSingle 'v', CApp (CSingle 'p', CApp (CSingle 'r', CSingle 'p')))` 1 pt

v Exercise 3: append Completed, 12 pts

Found `append` with compatible type.

Computing `append CEmpty (CApp (CEmpty, CApp (CEmpty, CSingle 4)))`

Correct value `(CApp (CEmpty, CApp (CEmpty, CSingle 4)))` 1 pt

Computing

```
append
  (CApp (CApp (CEmpty, CEmpty),
    CApp (CEmpty, CApp (CSingle (-4), CSingle 0))))
  (CApp (CSingle (-2), CEmpty))
```

Correct value 1 pt

```
(CApp
  (CApp (CApp (CEmpty, CEmpty),
    CApp (CEmpty, CApp (CSingle (-4), CSingle 0))),
  CApp (CSingle (-2), CEmpty)))
```

Computing

```
append
  (CApp (CApp (CSingle 2, CApp (CEmpty, CEmpty)),
    CApp (CSingle (-4), CEmpty)))
  (CApp (CApp (CApp (CSingle 3, CSingle (-1)), CSingle 2),
    CApp (CEmpty, CEmpty)))
```

Correct value 1 pt

```
(CApp
  (CApp (CApp (CSingle 2, CApp (CEmpty, CEmpty)),
    CApp (CSingle (-4), CEmpty)),
  CApp (CApp (CApp (CSingle 3, CSingle (-1)), CSingle 2),
   CApp (CEmpty, CEmpty))))
```

Computing `append (CSingle 0) (CApp (CEmpty, CSingle (-1)))`

Correct value `(CApp (CSingle 0, CApp (CEmpty, CSingle (-1))))` 1 pt

Computing

```
append
  (CApp
    (CApp (CApp (CSingle (-3), CSingle (-1)), CApp (CSingle 3, CSingle (-1))),
    CApp (CApp (CEmpty, CSingle (-4)), CApp (CSingle 1, CEmpty))))
  (CApp (CApp (CSingle (-2), CEmpty), CSingle (-3)))
```

Correct value 1 pt

```
(CApp
  (CApp
    (CApp (CApp (CSingle (-3), CSingle (-1)), CApp (CSingle 3, CSingle (-1))),
    CApp (CApp (CEmpty, CSingle (-4)), CApp (CSingle 1, CEmpty))),
  CApp (CApp (CSingle (-2), CEmpty), CSingle (-3))))
```

Computing `append (CSingle 1) (CApp (CEmpty, CEmpty))`

Correct value `(CApp (CSingle 1, CApp (CEmpty, CEmpty)))` 1 pt

Found `append` with compatible type.

Computing `append (CSingle 'w') CEmpty`

Correct value `(CSingle 'w')` 1 pt

Computing

```
append
  (CApp (CApp (CSingle 'x', CApp (CEmpty, CSingle 'x')),
    CApp (CEmpty, CSingle 'w')))
  (CSingle 'u')
```

Correct value 1 pt

```
(CApp
  (CApp (CApp (CSingle 'x', CApp (CEmpty, CSingle 'x')),
    CApp (CEmpty, CSingle 'w')),
  CSingle 'u'))
```

Computing `append (CSingle 'h') (CSingle 'n')`

```
    (CSingle 'm')
    (CApp (CSingle 't', CApp (CApp (CSingle 'u', CSingle 's'), CEmpty)))
```

Correct value 1 pt

```
  (CApp (CSingle 'm',
    CApp (CSingle 't', CApp (CApp (CSingle 'u', CSingle 's'), CEmpty))))
```

Computing

```
  append
    (CApp (CEmpty, CApp (CEmpty, CEmpty)))
    (CApp (CSingle 'd', CApp (CSingle 'y', CApp (CSingle 'i', CEmpty))))
```

Correct value 1 pt

```
  (CApp (CApp (CEmpty, CApp (CEmpty, CEmpty)),
    CApp (CSingle 'd', CApp (CSingle 'y', CApp (CSingle 'i', CEmpty)))))
```

Computing

```
  append
    (CApp (CApp (CSingle 'l', CApp (CEmpty, CSingle 'w')), CSingle 's'))
    (CApp (CSingle 'm', CSingle 'y'))
```

Correct value 1 pt

```
  (CApp (CApp (CApp (CSingle 'l', CApp (CEmpty, CSingle 'w')), CSingle 's'),
    CApp (CSingle 'm', CSingle 'y')))
```

v Exercise 4: hd Completed, 11 pts

Found hd with compatible type.

Computing

```
  hd
    (CApp
      (CApp (CApp (CSingle (-4), CEmpty), CApp (CSingle (-1), CSingle (-4))),
      CApp (CSingle 4, CSingle (-1))))
```

Correct value (Some (-4)) 1 pt

Computing hd (CApp (CSingle (-5), CEmpty))

Correct value (Some (-5)) 1 pt

Computing

```
  hd
    (CApp (CApp (CApp (CSingle 4, CSingle (-2)), CApp (CSingle (-3), CEmpty)),
      CEmpty))
```

Correct value (Some 4) 1 pt

Computing hd (CSingle 1)

Correct value (Some 1) 1 pt

Computing

```
  hd
    (CApp (CApp (CApp (CSingle 4, CSingle 1), CEmpty),
      CApp (CSingle (-1), CEmpty)))
```

Correct value (Some 4) 1 pt

Found hd with compatible type.

Computing hd CEmpty

Correct value None 1 pt

Computing hd (CApp (CApp (CSingle 'q', CApp (CEmpty, CSingle 'x')), CSingle 'o'))

Correct value (Some 'q') 1 pt

Computing hd (CSingle 'h')

Correct value (Some 'h') 1 pt

Computing hd (CApp (CApp (CSingle 'g', CSingle 't'), CSingle 'y'))

Correct value (Some 'g') 1 pt

Computing hd (CApp (CSingle 'j', CApp (CEmpty, CSingle 't')))

Correct value (Some 'j') 1 pt

Computing

```
  hd
    (CApp (CApp (CApp (CSingle 'p', CSingle 'w'), CApp (CEmpty, CEmpty)),
      CApp (CSingle 'j', CSingle 'x')))
```

Correct value (Some 'p') 1 pt

v Exercise 5: tl Completed, 11 pts

Found tl with compatible type.

Computing tl (CSingle 4)

Correct value (Some CEmpty) 1 pt

Computing tl (CApp (CEmpty, CEmpty))

Correct value None 1 pt

Computing tl (CApp (CApp (CSingle 2, CEmpty), CSingle (-4)))

Correct value (Some (CSingle (-4))) 1 pt

Computing tl (CApp (CSingle (-4), CSingle (-4)))

Correct value (Some (CSingle (-4))) 1 pt

Computing tl (CSingle (-5))

Correct value (Some CEmpty) 1 pt

Found tl with compatible type.

Computing tl CEmpty

Correct value None 1 pt

Correct value `(Some CEmpty)`	1 pt
Computing `tl (CSingle 'e')`	
Correct value `(Some CEmpty)`	1 pt
Computing `tl (CSingle 'y')`	
Correct value `(Some CEmpty)`	1 pt
Computing `tl (CApp (CEmpty, CSingle 's'))`	
Correct value `(Some CEmpty)`	1 pt