CS 496: Quiz 4C

30 March, 2022

Exercise 1

Implement a recursive function `take` by completing the REC source file `take.rec` provided in the stub. It should behave as follows: `((take n) l)` should return the list resulting from taking the first `n` elements from the list `l`. If `n` is larger than the size of `l`, then it returns `l` itself. Also, if `n` is 0 then it returns the empty list.

Complete the stub provided:

```
letrec take(n) = (* complete *)
in let example =  cons(1,cons(2,cons(3,cons(4,emptylist))))
in ((take 2) example)
```

The list operations that are available are the following (with the obvious meanings): `emptylist`, `cons(e1,e2)`, `hd(e)`, `tl(e)`, and `empty?(e)`.

Here is how you may run your code:

```
utop # interpf "take";;
- : exp_val Rec.Ds.result = Ok (ListVal [NumVal 1; NumVal 2])
```

Exercise 2

Implement a recursive function that reverses a list by completing the following REC source file (see file `rev.rec` provided in the stub).

```
letrec rev_helper = (* your code here *)
in let rev = (* your code here *)
in let example = cons(1,cons(2,cons(3,cons(4,emptylist))))
in (rev example)
```

The `rev` function is not declared recursive since its implementation will rely on a helper function, which will be recursive and should be declared before `rev`, that you must supply.

Once completed, evaluation of the code above in utop should produce

```
utop # interpf "rev";;
- : exp_val Rec.Ds.result = Ok (ListVal [NumVal 4; NumVal 3; NumVal 2; NumVal 1])
```

Hint: use tail-recursion.