CS 2041: Practice Exam 2 SOLUTION

Fall 2018
University of Minnesota

Exam period: 20 minutes
Points available: 40

Problem 1 (10 pts): Write a function called `strlen_betwen` which accepts a min and max length and a list of strings. A list of strings with length in the given range is returned. Min/max lengths are inclusive. For full credit, make use of a **h**igher-order function in your definition. Demo uses are given below.

```
# #use "strlen_between.ml";;
val strlen_between :
int -> int -> string list -> string list = <fun>
# let lst = ["aaa";"bbbbb"; "cccc";"dddddd"];;
val lst : string list =
["aaa"; "bbbbb"; "cccc"; "dddddd"]
# strlen_between 2 4 lst;;
- : string list = ["aaa"; "cccc"]
# strlen_between 4 7 lst;;
- : string list = ["bbbbb"; "cccc"; "dddddd"]
# strlen_between 9 12 lst;;
- : string list = []
```

SOLUTION:

```
let strlen_between min max =
  let filt s =
    let len = String.length s in
    min <= len && len <= max
  in
  List.filter filt
;;
```

Problem 2 (10 pts): Write a function called `largest_even` which accepts a list of integers. If no even numbers are in the list, return `None`. Otherwise, return `Some` of the largest integer in the list. For full credit, use pattern matching and a higher-order function in your solution. Demo uses are given below.

```
# #use "largest_even.ml";;
val largest_even :
  int list -> int option = <fun>
# largest_even [];;
- : int option = None
# largest_even [1;3;5];;
- : int option = None
# largest_even [4];;
- : int option = Some 4
# largest_even [1;3;2;5];;
- : int option = Some 2
# largest_even [1;4;3;2;5;8;7;6];;
- : int option = Some 8
```

SOLUTION:

```
let largest_even list =
  let help iopt x =
    match iopt,x with
    | None,x when x mod 2=0  ->
       Some x
    | (Some e),x when x mod 2 = 0 && x > e ->
       Some x
    | _ ->
       iopt
  in
  List.fold_left help None list
;;

(* Several alternate solutions will be posted
   on the Piazza including those discussed in
   lecture.
*)
```

Background: Shae Lowcopy decided to extend the `multimanager` application to allow the current list to be copied using the `copyto` command. She modifies `multimanager.ml` to include the following additional command sequence.

```
let execute_command tokens =
  let cmd = tokens.(0) in       (* 0th element is command *)
  match cmd with
  ...
  | "copyto" ->
     let new_name = tokens.(1) in
     let added = Doccol.add global new_name global.curdoc in
     if added then
       printf "Copied list to '%s'\n" new_name
     else
       printf "ERROR: list '%s' already exists, cannot create copy\n" new_name
  ...
```

Unfortunately when she begins testing she sees the below undesirable behavior: the copied list seems to affect the original list when they should be independent.

```
> multimanager
(default.txt)> add Korra
(default.txt)> add Mako
(default.txt)> add Bolin
(default.txt)> show
--BEG LIST--
Bolin
Korra
Mako
--END LIST--

(default.txt)> copyto others.txt
Copied list to 'others.txt'
(default.txt)> lists
2 docs
- others.txt
- default.txt

(default.txt)> edit others.txt

(others.txt)> show
--BEG LIST--
Bolin
Korra
Mako
--END LIST--

(others.txt)> remove Korra
(others.txt)> add Asami
(others.txt)> showall
--List others.txt--
Asami
Bolin
Mako

--List default.txt--
Asami
Bolin
Mako
```

Problem 3 (10 pts): Describe in some detail why Shae's implementation of `copyto` does not work as expected and why the two lists seem to be linked somehow.

Shae is only making a shallow copy of an existing doc. That means other functions that mutate the fields of the doc to add and remove elements will affect the original version. In the example, the names default.txt and others.txt both refer to the same document.

Problem 4 (10 pts): Describe how to fix the problem so that a proper list copy is made for `copyto`. Provide at least some code to give a concrete idea of how your idea would work.

Create a copy of the document record with the same current list. This can done with a call to `Document.make` *to initialize the copied document state to the current doc. This solution does not include the undo/redo history which would need to be copied as well if it were to be preserved. If the documents tracked a mutable kind, not* `string list`*, then a deeper copy would be necessary to avoid sharing.*

```
  | "copyto" ->
     begin
       let new_name = tokens.(1) in
       let new_doc =
         Document.make global.curdoc.current in
       let added =
         Doccol.add global new_name new_doc in
       if added then
         printf "Copied list to '%s'\n" new_name
       else
         printf "ERROR: list '%s' already exists"
           new_name
     end
```