Problem 1 (10 pts): Write a function called `even_indices` which takes any type of list and returns a list of elements at even indices 0, 2, 4, etc. Example uses from a REPL are shown. **Hint:** a recursive solution which “skips” element is effective. My if/else solution is 13 lines long while pattern matching makes this considerably shorter.

```ocaml
let rec even_indices list =
  if list=[] then
    []
  else
    let head = List.hd list in
    let tail = List.tl list in
    if tail=[] then
      [head]
    else
      let rest = even_indices (List.tl tail) in
      head::rest
  ;;

let rec even_indices list =
  match list with
  | [] -> []
  | head :: [] -> [head]
  | even :: odd :: tail -> even :: (even_indices tail)
  ;;
```

Problem 2 (10 pts): Source code for the `array_fill` function is provided along with a short session which attempts to demonstrate the function. A warning is given on loading the code and an unexpected result occurs. Describe the following.

(A) Why is the warning given?  
**SOLUTION:** Line 3 of the function is not doing array assignment but an equality check instead. The warning is indicating that a boolean results rather than a unit which is expected for assignment.

(B) Why is the array apparently unchanged?  
**SOLUTION:** Since `elem` is not actually assigned to any array elements, the array remains unchanged.

(C) How can the function be corrected to remove the warning and carry out its intended purpose? 
**SOLUTION:** Simply replace the `=` sign on line 3 with the array assignment operator `<-`; this has return type `unit` and will actually change elements of the array.

> cat -n fill.ml
1 (* fill array with given element *)
2 let fill_array arr elem =
3   for i=0 to (Array.length arr)-1 do
4     arr.(i) = elem;
5   done;
6 ;;

> ocaml
# #use "fill.ml";;
File "fill.ml", line 3, characters 4-18:
Warning 10: this expression should have type unit.
val fill_array : 'a array -> 'a -> unit = <fun>

# let a = [|9;5;2|];;
val a : int array = [|9; 5; 2|]

# fill_array a 7;;
- : unit = ()
# a;;
- : int array = [|9; 5; 2|]
Problem 3 (10 pts): Complete the pointer diagram to shown to reflect how the OCaml code will use existing cons boxes and create new ones.

SOLUTION:

```ocaml
let listX = [6; 5; 1];;

let ansA = List.hd (List.tl listX);;

let ansB = List.tl (List.tl listX);;

let ansC = 7 :: listX;;

let ansD = 9 :: 5 :: ansB;;
```

Problem 4 (10 pts): Write a function called `firstlast` which returns a list of the first and last elements of a parameter list. For empty lists, the empty list is returned. For single element lists, only that element is returned. For full credit, make use of a tail-recursive helper function to complete the function.

Many solutions are possible, 2 are shown below that use tail recursive helper functions.

SOLUTION 1:

```ocaml
1 let firstlast list =
2   if list=[] then
3     []
4   else
5     let first = List.hd list in
6     let rest = List.tl list in
7     if rest=[] then
8       [first]
9     else
10    let rec helper lst =
11       let head = List.hd lst in
12       let tail = List.tl lst in
13       if tail=[] then
14         [first; head]
15       else
16         helper tail
17      in
18     helper rest
19   ;;
```

SOLUTION 2:

```ocaml
1 (* REPL demo for firstlast *)
2 # firstlast [];;
3 - : 'a list = []
4 # firstlast ["a"];;
5 - : string list = ["a"]
6 # firstlast ["a"; "b"];;
7 - : string list = ["a"; "b"]
8 # firstlast ["a"; "b"; "c"; "d"];;
9 - : string list = ["a"; "d"]
10 # firstlast ["a"; "b"; "c"; "d"; "e"; "f"];;
11 - : string list = ["a"; "f"]
12 # firstlast [1;2;3;4;5;6];;
13 - : int list = [1; 6]
```

SOLUTION 2:

```ocaml
1 (* pattern matching version *)
2 let firstlast list =
3   match list with
4     | [] -> []
5     | first :: [] -> [first]
6     | first :: rest ->
7     | last :: [] -> [first; last]
8     | head :: tail -> helper tail
9     | _ -> failwith "Something's wrong"
10    (* last case avoids compile warning *)
11   in
12  helper rest
13 ;;
```