프로그래밍 언어 개론
과제 보고서

[PL00]HW4_201601980_KimSeongHan

개발환경 : Ubuntu

제출일 (2021-03-30)

201601980 / 김성한(00분반)

<과제 설명>

Homework

```
type t = (string * string)list
```

List와 tuple을 활용한 Map을 구현하시오.

- let empty = []
- let add key value map = (* write your code *)
  - string -> string -> t -> t
- let rec find key map = (* write your code *)
  - string -> t -> string
  - key가 없으면, failwith "No such key exists"
- let erase key map = (* write your code *)
  - string -> t -> t
  - key가 없으면, failwith "No such key exists"

에러메시지 꼭 지켜주세요. 오타 불허.

[문제 HomeWork]

● 이번 과제는 List 및 tuple을 활용하여 Map을 구현하는 과제로 Map 자료구조의 메소드인 add find erase 세가지 메소드를 구현하는 과제이다.

● data type으로는 (string*string) string 원소 두 개로 이루어진 튜플 리스트가 정의되었으며, add 함수는 두 개의 string 인자와 t타입((string * string)list)의 인자를 받아 인자로 들어온 map 리스트에 해당 (key,value) 튜플을 추가해준다. find 함수는 key 와 map을 인자로 받아 해당 map에서 key가 같은 튜플에 대한 value를 반환한다. 마지막으로 erase 함수는 key 와 map을 인자로 받고 해당 key에 대한 튜플을 리스트에서 제거한다.

● 위 내용을 바탕으로 함수 작성을 시작하였으며 함수의 타입에 최대한 신경을 쓰면서 함수를 작성하였다.

<코드 구현 및 실행 결과 화면>

[제공된 코드 1]

● 초기 Format을 F 로 정의해주었고, (string * string) 형태의 튜플을 가지는 list (string * string)list를 type t로 정의하였다. empty 에는 빈 리스트를 할당하였다.

```
let add key value map =
        ((key,value) :: map)
```

[listMap.ml 내부의 add 함수의 코드]

● add 함수는 key(string) value(string) map(t) 의 인자를 받으며, 인자로 들어온 map 리스트에 (key,value) 튜플을 추가한다. 이때 map의 앞부분에 튜플을 추가해주기 위해 :: 연산자를 이용해 추가해 주었다.

```
let rec find key map =
        match map with
        |[] -> failwith "No such key exists"
        |(k,v) :: t -> if key = k then v else (find key t)
```

[listMap.ml 내부의 find 함수의 코드]

● find 함수는 key(string) map(t) 의 인자를 받으며 map을 비교하여 해당 튜플의 키가 인자로 들어온 key와 같으면 해당 value(string)을 반환한다. 아닌 경우 나머지 리스트에 대하여 find를 재귀적으로 반복한다. 이렇게 반복하다 [] 가 될 때까지 진행한다면 해당 리스트내에 key와 같은 키를 가진 튜플은 존재하지 않는다는 의미이고 failwith을 이용해 오류문 “No such key exists”을 출력해준다.

```
let erase key map =
        let l = List.length map in
        let rec erase_2 key map acc =
                match map with
                |[] -> acc
                |(k,v)::t -> if key = k then (erase_2 key t acc)
                             else (erase_2 key t (acc @ [(k,v);]))
        in
        let newMap =  erase_2 key map [] in
        if List.length newMap = l then failwith "No such key exists" else newMap
```

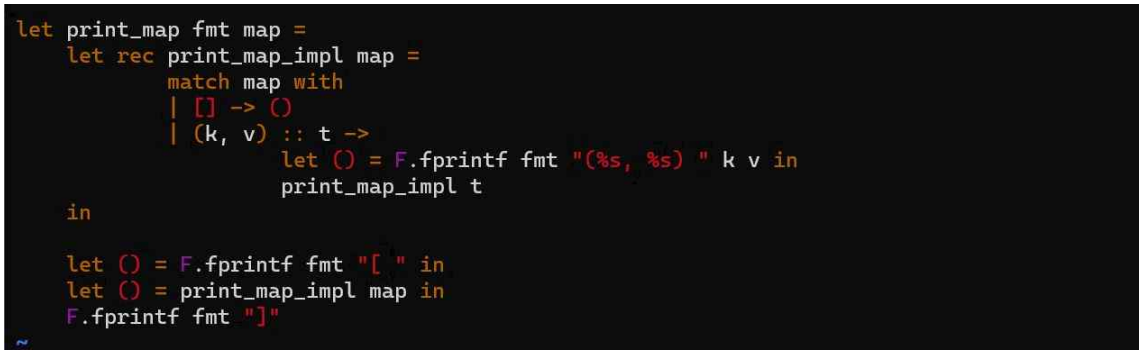
[listMap.ml 내부의 erase 함수의 코드]

● erase 함수는 key(string) map(t)를 인자로 가지며 주어진 코드와 다르게 해당 함수를 래퍼함수로 변형해주었다. 초기 변수 l에 List.length map (<- map 리스트의 길이)를 할당해주었다. 이는 초기 인자로 들어온 map 리스트의 길이를 기억하기 위함이다.

● 그 후 내부에 tail-recursive 하게 erase_2함수를 구현하였다. erase_2함수는 래퍼함수의 인자에서 추가적으로 acc 인자를 하나 더 가지고 있다. acc인자는 비교한 key 값이 다를 경우 acc에 접합해줌으로써 다른 key를 가지는 리스트를 새로 생성하여 다음 재귀문의 인자로 들어간다. patter-matching을 이용하여 map 리스트가 만일 빈 리스트인 경우 acc를 반환하고, 아닌 경우 (k,v)::t를 통해 map으로부터 맨앞의 튜플을 분리한다. 이렇게 분리된 튜플의 k 와 인자로 들어온 key를 비교하여 만일 같다면 남은 리스트 t에 대해 다시 재귀를 돌려준다. 이때 acc에는 아무 원소도 추가하지 않는다. 반면 key 와 k 가 다를 경우 재귀를 돌리는데 이 과정에서 acc에 (k,v) 튜플을 추가해준다.

● 내부에서 재귀적으로 돌아가는 함수이기에 in을 붙여준다. 그 후 바로 밑줄에서 해당함수를 호출하는데 호출하고 반환되게 되는 acc 리스트를 newMap 변수에 할당해준다. 이때 초기 acc의 값은 [] 빈리스트가 인자로 들어가게 된다.

● 그 후 if-then-else 문을 이용하여 만일 초기 map에 대해 erase를 하기 전 길이 l 과 newMap의 길이가 같다면 map 리스트에는 제거하려한 key와 같은 key를 가진 원소가 없다는 의미이므로 failwith을 이용하여 “No such key exists”를 출력해준다. 아닌 경우에는 erase가 진행된 newMap리스트를 반환한다.

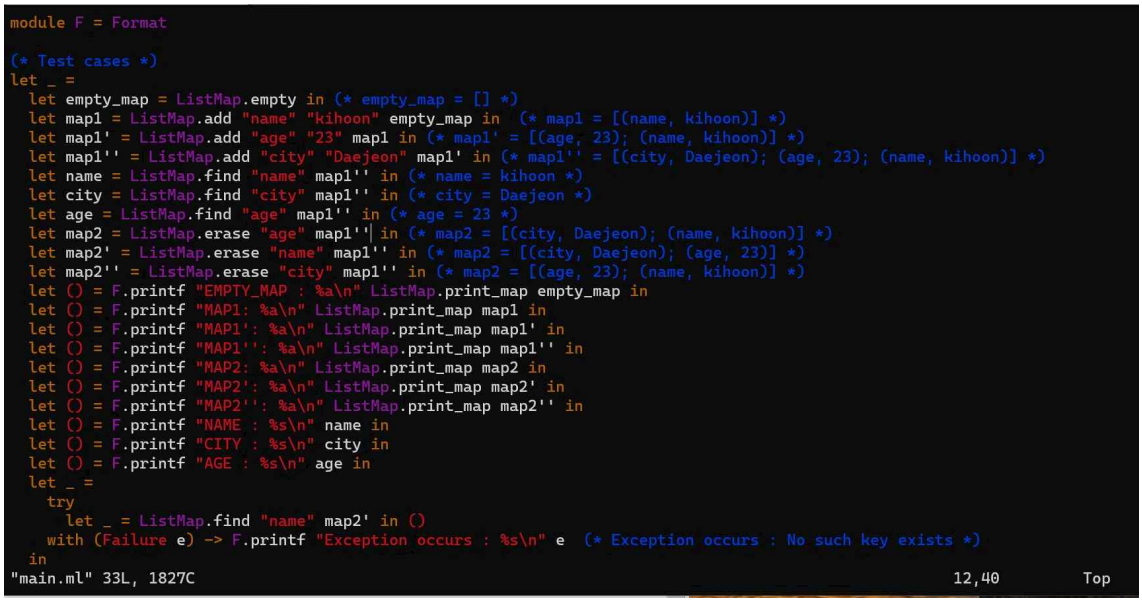


```
let print_map fmt map =
    let rec print_map_impl map =
            match map with
            | [] -> ()
            | (k, v) :: t ->
                    let () = F.fprintf fmt "(%s, %s) " k v in
                    print_map_impl t
    in

    let () = F.fprintf fmt "[ " in
    let () = print_map_impl map in
    F.fprintf fmt "]"
```

[제공된 코드 2]

● print_map fmt map 함수로 map 리스트의 내용을 보기 좋게 출력해주기 위한 역할을 한다. 초기 데이터 타입 및 변수 및 모듈선언 부분과 함께 제공된 코드이다.

● 인자로 fmt map을 받으며 인자로 들어온 map에 대해 [] 괄호를 씌우고 튜플들을 묶어 출력해준다.



```
module F = Format

(* Test cases *)
let _ =
  let empty_map = ListMap.empty in (* empty_map = [] *)
  let map1 = ListMap.add "name" "kihoon" empty_map in  (* map1 = [(name, kihoon)] *)
  let map1' = ListMap.add "age" "23" map1 in (* map1' = [(age, 23); (name, kihoon)] *)
  let map1'' = ListMap.add "city" "Daejeon" map1' in (* map1'' = [(city, Daejeon); (age, 23); (name, kihoon)] *)
  let name = ListMap.find "name" map1'' in (* name = kihoon *)
  let city = ListMap.find "city" map1'' in (* city = Daejeon *)
  let age = ListMap.find "age" map1'' in (* age = 23 *)
  let map2 = ListMap.erase "age" map1'' in (* map2 = [(city, Daejeon); (name, kihoon)] *)
  let map2' = ListMap.erase "name" map1'' in (* map2 = [(city, Daejeon); (age, 23)] *)
  let map2'' = ListMap.erase "city" map1'' in (* map2 = [(age, 23); (name, kihoon)] *)
  let () = F.printf "EMPTY_MAP : %a\n" ListMap.print_map empty_map in
  let () = F.printf "MAP1: %a\n" ListMap.print_map map1 in
  let () = F.printf "MAP1': %a\n" ListMap.print_map map1' in
  let () = F.printf "MAP1'': %a\n" ListMap.print_map map1'' in
  let () = F.printf "MAP2: %a\n" ListMap.print_map map2 in
  let () = F.printf "MAP2': %a\n" ListMap.print_map map2' in
  let () = F.printf "MAP2'': %a\n" ListMap.print_map map2'' in
  let () = F.printf "NAME : %s\n" name in
  let () = F.printf "CITY : %s\n" city in
  let () = F.printf "AGE : %s\n" age in
  let _ =
    try
      let _ = ListMap.find "name" map2' in ()
    with (Failure e) -> F.printf "Exception occurs : %s\n" e  (* Exception occurs : No such key exists *)
  in
"main.ml" 33L, 1827C                                                    12,40         Top
```

[제공된 main.ml]

```
k0s0a7@DESKTOP-MLPLCFD:~/week4/hw$ dune exec ./main.exe
EMPTY_MAP : [ ]
MAP1: [ (name, kihoon) ]
MAP1': [ (age, 23) (name, kihoon) ]
MAP1'': [ (city, Daejeon) (age, 23) (name, kihoon) ]
MAP2: [ (city, Daejeon) (name, kihoon) ]
MAP2': [ (city, Daejeon) (age, 23) ]
MAP2'': [ (age, 23) (name, kihoon) ]
NAME : kihoon
CITY : Daejeon
AGE : 23
Exception occurs : No such key exists
Exception occurs : No such key exists
```

[제공된 main 문을 실행한 결과]

● main 문을 보면 초기 빈 리스트에 대해 튜플 3개를 추가해주고, find 메소드를 이용하여 해당 튜플들을 찾아 주었다. 그 후 erase를 통해 각각 erase한 경우에 대한 결과를 출력하고 있으며, 제시된 결과에 대해 옳은 결과를 출력해 줌을 알 수 있다.

<과제 구현 중 어려웠던 부분 및 해결 과정>

```
let rec erase key map =
        match map with
        |[] -> failwith "No such key exists"
        |(k,v) :: t -> if key <> k then ( (k,v) :: erase key t) else t
```

[erase 함수를 tail-recursive 하게 구현하기 이전 코드]

● 이번 과제에서 가장 어려웠던 부분을 뽑자면 erase 함수를 고를 수 있다.

● 초기 작성한 방식으로는 tail-recursive하게 함수를 작성하지 않았었다. 위의 사진에 있는 코드를 보면 인자로 들어온 map 리스트에 대하여 빈 리스트일 경우 에러를 출력해주며, 아닌 경우 튜플을 하나 분리하여 만일 해당 튜플의 키가 인자로 들어온 key와 다를 때 해당 튜플을 erase key t 의 재귀 결과에 추가를 해주었다. 만일 같은 경우에 대해서는 tail을 반환하였는데, 이 코드에는 심각한 오류가 있었다. 만일 key 가 같은경우가 나온다면 tail이 반환되고 add 구현특성상 같은 key를 가진 튜플이 있을 수 있기에 tail에 만일 찾는 key와 같은 튜플이 남아 있을 경우 제거되지 않고 반환되게 된다. 이 과정에서 오류를 감지하지 못하고 무작정 testcase의 결과가 나왔다고 좋아했었는데 정말 창피했다.

● 따라서 위의 코드를 수정하였고, 보고서를 쓰던 중 tail_recursive 하지 않게 코드를 작성할 수 있는 방법이 문득 떠올랐다. 아래와 같이 작성할 수 있었다.

```
let erase key map =
        (*erase 함수 : not tail-recursion*)
        let l = List.length map in
        let rec erase_2 key map =
                match map with
                |[] -> []
                |(k,v) :: t -> if key = k then erase_2 key t
                               else ((k,v) :: erase_2 key t)
        in
        let newMap = erase_2 key map in
        if List.length newMap = l then failwith "No such key exists" else newMap
```

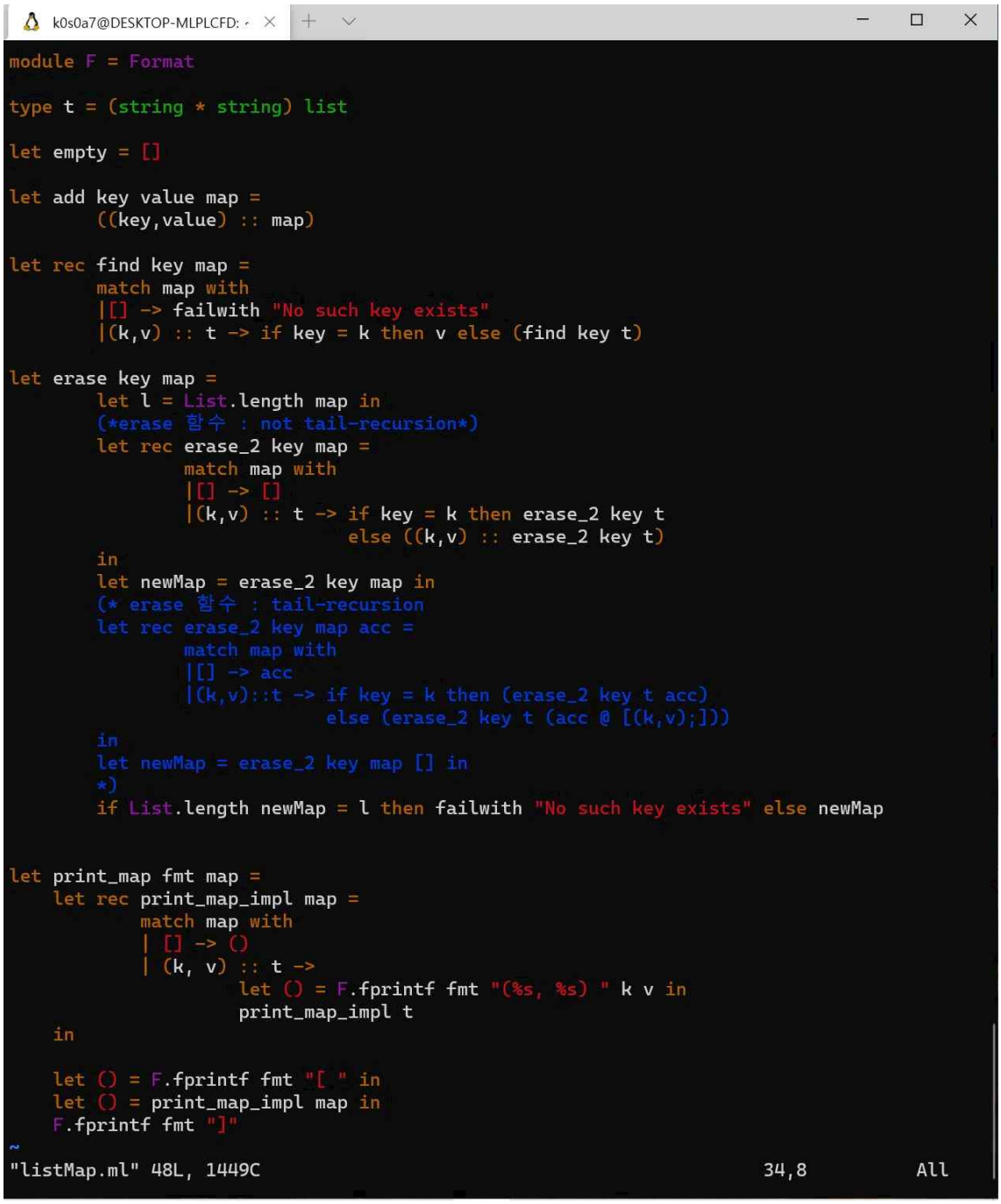
[erase 함수를 tail-recursive 하지 않게 오류 없이 수정한 모습]

● 너무 갑작스럽게 떠올랐지만 tail-recursive하게 함수를 구현하고 나서야 tail-recursive하지 않게 함수를 구현할 방법이 생각이 났다. 위의 코드는 앞에서 설명한 tail-recursive한 erase함수와 비슷하게 동작하며 pattern-matching 부분만 차이가 있다.

● (k,v) 튜플 한 개를 인자 map으로부터 분리하여 만일 k와 key가 같다면 다시 erase_2 key t를 호출한다. 만일 key가 다르다면 해당 튜플을 :: 연산을 이용하여 재귀호출된 함수로부터 반환된 리스트에 해당 튜플을 삽입한다.

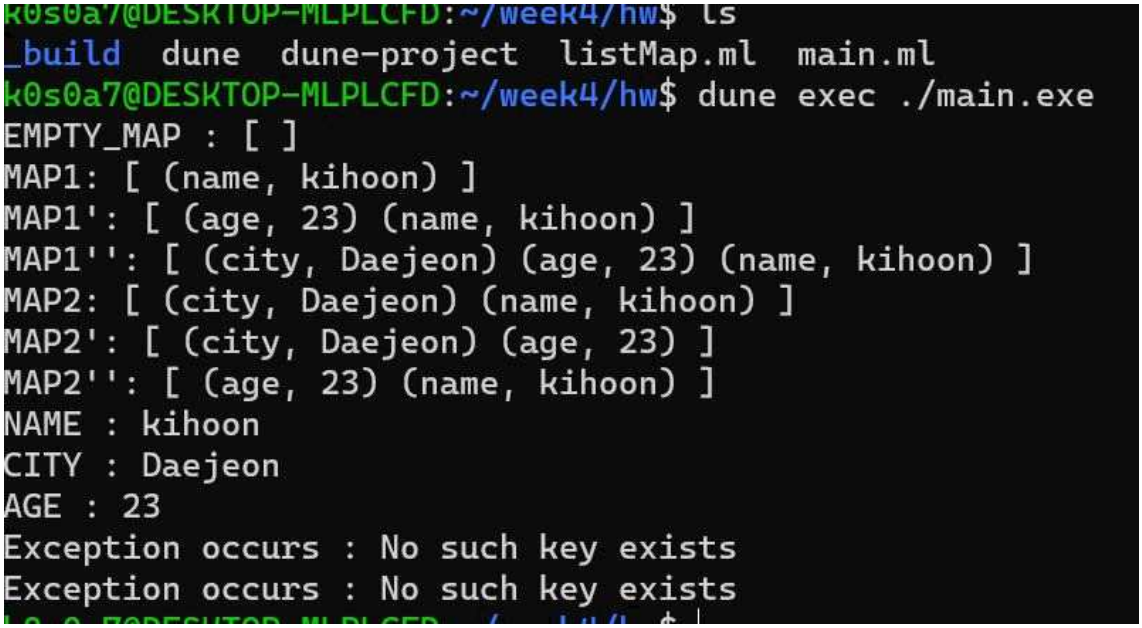
● 구현 후 결과를 확인하였고, 테스트 케이스도 조금씩 바꾸어 실행해보니 확신이 들었다.

<최종 제출 코드 및 결과 화면>



```
k0s0a7@DESKTOP-MLPLCFD:
module F = Format

type t = (string * string) list

let empty = []

let add key value map =
        ((key,value) :: map)

let rec find key map =
        match map with
        |[] -> failwith "No such key exists"
        |(k,v) :: t -> if key = k then v else (find key t)

let erase key map =
        let l = List.length map in
        (*erase 함수 : not tail-recursion*)
        let rec erase_2 key map =
                match map with
                |[] -> []
                |(k,v) :: t -> if key = k then erase_2 key t
                               else ((k,v) :: erase_2 key t)
        in
        let newMap = erase_2 key map in
        (* erase 함수 : tail-recursion
        let rec erase_2 key map acc =
                match map with
                |[] -> acc
                |(k,v)::t -> if key = k then (erase_2 key t acc)
                             else (erase_2 key t (acc @ [(k,v);]))
        in
        let newMap = erase_2 key map [] in
        *)
        if List.length newMap = l then failwith "No such key exists" else newMap


let print_map fmt map =
    let rec print_map_impl map =
            match map with
            | [] -> ()
            | (k, v) :: t ->
                    let () = F.fprintf fmt "(%s, %s) " k v in
                    print_map_impl t
    in

    let () = F.fprintf fmt "[ " in
    let () = print_map_impl map in
    F.fprintf fmt "]"
~
"listMap.ml" 48L, 1449C                                          34,8          All
```

[erase의 두가지 구현을 포함(tail-recursion은 주석처리 하였습니다.)한 list.ml 파일]



[list.ml 파일의 함수를 main.exe를 실행하여 확인한 모습]

● 초기 제출하려 했던 코드와 다르게 보고서 작성 중 tail-recursive하지 않게 구현한 erase함수가 떠올라 이를 주로 하여 기존에 tail-recursive 한 erase는 주석처리하여 제출코드로 결정하였습니다.

● 두 번째 사진은 다시 해당 코드를 실행한 결과이며 오류가 없음을 알 수 있습니다.

<새로 고찰한 내용 및 느낀점>

● 이번 과제를 하면서 오랜만에 좋은 느낌을 받은 것 같다. 초기 add 함수와 find 함수를 구현함에 있어서 어려운 점은 크게 없었지만 erase함수를 구현하면서는 너무 생각이 갇혀있었다. pattern을 matching 하는 부분에서 모든 처리를 다해주려고 단편적으로 생각하다보니 생긴 일인 것 같다. 아무리 생각해도 tail-recursive하게 구현하는 방법밖에는 떠오르지 않아 결국 tail-recursive 하게 작성하였지만 교수님께서 실습시간 도중 tail-recursive하게 작성하지 않을 수 있다고 하신 말씀이 마음에 걸렸다. 결국 포기하고 보고서를 작성하는 도중 가장 어려웠던 부분이었던 erase 함수에 대해 작성하려 했을 때 이전에 썼던 코드를 가만히 보고 있으니 문득 구현이 가능할 것 같다는 느낌을 받았고 바로 터미널 창으로 옮겨 작성한 뒤에는 너무 기분이 좋았다. 그래서 기존 작성한 코드는 주석처리를 하고 not tail-recursive 한 함수를 실행 가능하도록 작성하였다. 이외에도 Ocaml에서 List의 길이를 구하는 List.length 함수의 사용법 및 지난주에 이어 다시 한번 래퍼함수를 이용하여 구현하면서 점차 Ocaml의 구문과 메소드가 더 머릿속에 들어오고 있음을 느꼈다. 많이 부족하지만 앞으로도 열심히 해야겠다.