Ozyegin University
CS 321 Programming Languages
Sample Problems on Streams

Ee use the following definition of streams:

```
type 'a stream = Cons of 'a * (unit -> 'a stream)
```

1. Define the stream of square numbers. That is, the stream of 1, 4, 9, 16, 25, 36, . . .

Solution:

```
let rec squaresFrom n = Cons(n*n, fun() -> squaresFrom (n+1))

let squares = squaresFrom 1
```

2. Implement a `cycle` function that takes a list `lst` and converts it to an infinite stream as if `lst` were a circular list.

```
# cycle;;
val cycle : 'a list -> 'a stream = <fun>
# let letters = cycle ['a'; 'b'; 'c'; 'd'];;
val letters : char stream = Cons ('a', <fun>)
# take 15 letters;;
- : char list =
['a'; 'b'; 'c'; 'd'; 'a'; 'b'; 'c'; 'd'; 'a'; 'b'; 'c'; 'd'; 'a'; 'b'; 'c']
# take 10 (cycle ['a']);;
- : char list = ['a'; 'a'; 'a'; 'a'; 'a'; 'a'; 'a'; 'a'; 'a'; 'a']
```

Solution:

```
let cycle lst =
  let rec helper items =
    match items with
    | [] -> helper lst
    | x::xs -> Cons(x, fun () -> helper xs)
  in helper lst
```

3. Implement the `takeWhile` function for streams that takes a predicate `p`, a stream `s`, and returns as a list all the elements of `s` until there is an element of `s` that does not satisfy `p`. **Extra: give an efficient tail-recursive solution.**

```
# takeWhile (fun n -> n < 100) squares;;
- : int list = [1; 4; 9; 16; 25; 36; 49; 64; 81]
```

Solution:

```
let takeWhile p st =
  let rec helper st acc =
    let x = head st
    in if p x then helper (tail st) (x::acc) else acc
  in List.rev(helper st [])
```

4. Write an OCaml function `enumerate` that takes a stream and enumerates its elements starting from 0.

```
# enumerate;;
- : 'a stream -> (int * 'a) stream = <fun>
# let letters = cycle ['e'; 'n'; 'u'; 'm'];;
val letters : char stream = Cons ('a', <fun>)
# take 10 letters;;
- : char list = ['e'; 'n'; 'u'; 'm'; 'e'; 'n'; 'u'; 'm'; 'e'; 'n']
# take 10 (enumerate letters);;
- : (int * char) list =
[(0, 'e'); (1, 'n'); (2, 'u'); (3, 'm'); (4, 'e'); (5, 'n'); (6, 'u');
  (7, 'm'); (8, 'e'); (9, 'n')]
```

Solution:

```
let enumerate st =
  let rec helper n st =
    Cons((n, head st), fun() -> helper (n+1) (tail st))
  in helper 0 st
```

5. Write an OCaml function named `merge` that combines two streams into one by taking elements alternately. (i.e. *bi ordan bi burdan*)

```
# merge;;
- : 'a stream -> 'a stream -> 'a stream = <fun>
# let evens = filter (fun n -> n mod 2 = 0) naturals;;
val evens : int stream = Cons (0, <fun>)
# take 10 evens;;
- : int list = [0; 2; 4; 6; 8; 10; 12; 14; 16; 18]
# let odds = filter (fun n -> n mod 2 = 1) naturals;;
val odds : int stream = Cons (1, <fun>)
# take 10 odds;;
- : int list = [1; 3; 5; 7; 9; 11; 13; 15; 17; 19]
# take 10 (merge odds evens);;
- : int list = [1; 0; 3; 2; 5; 4; 7; 6; 9; 8]
# take 10 (merge (cycle ['a']) (cycle ['b']));;
- : char list = ['a'; 'b'; 'a'; 'b'; 'a'; 'b'; 'a'; 'b'; 'a'; 'b']
```

Solution:

```
let rec merge st1 st2 =
  Cons(head st1, fun() ->
               Cons(head st2, fun() ->
                            merge (tail st1) (tail st2)))
```