演習問題 4.1

以下の2つ関数 `drop1` `drop2` の定義が等価であることを示す。

定義が等価とは？ → おそらく、任意の入力について出力が等しくなるような2つの関数が等価

```
fun lazy drop1 (0, s) = s
       | drop1 (n, $Nil) = $Nil
       | drop1 (n, $Cons(x, s)) = drop1 (n-1, s)
```

```
fun lazy drop2 (n, s) =
  let fun drop' (0, s) = s
        | drop' (n, $Nil) = $Nil
        | drop' (n, $Cons(x, s)) = drop' (n-1, s)
  in drop' (n, s) end
```

`drop1` を脱糖すると、

```
fun drop1 x = $case x of
  (0, s) => force s
  (n, $Nil) => force ($Nil)
  (n, $Cons(x, s)) => force (drop1 (n-1, s))
```

また、 `drop2` を脱糖すると、

```
fun drop2 x = $case x of
  (n, s) =>
    let fun drop' (0, s) = s
          | drop' (n, $Nil) = $Nil
          | drop' (n, $Cons(x, s)) = drop' (n-1, s)
    in force (drop' (n, s)) end
```

ここで、 `drop'` は正格なパターンしかマッチしないため、 `force` は `drop'` の適用後に対して働くことになる。

```
fun drop2 x = $case x of
  (n, s) =>
    let fun drop' (0, s) = force s
          | drop' (n, $Nil) = force ($Nil)
          | drop' (n, $Cons(x, s)) = force (drop' (n-1, s))
    in drop' (n, s) end
```

`x` に対するパターンマッチは `(n, s)` の1通りのみであり、マッチした結果はそのまま `drop'` に渡される。ここで、 `drop'` のパターンマッチの定義が、脱糖した `drop1` のcase式のパターンマッチの定義と等価であるため、 `drop2` の最終的な評価結果は、 `drop1` と常に一致することがわかる。

メモ: 停止計算が一枚岩とは

引数に取った `$x` の形式の `x` 部分を正格評価してから計算に使い、 `$` で包んで返す、みたいなことをしている

メモ: 糖衣構文

- `force($e)` ⟺ `e`
- `fun lazy f p = e` ⟺ `fun f x = $case x of p => force e`

メモ: 具体例への適用

長さ3のリストから1要素ドロップする例を考える

```
> drop1 (1, $Cons(1, $Cons(2, $Cons(3, $Nil))))
= $force ($drop1 (0, $Cons(2, $Cons(3, $Nil)))) -- drop1 パターン3 適用
= $drop1 (0, $Cons(2, $Cons(3, $Nil)))          -- force 適用
= $force ($Cons(2, $Cons(3, $Nil)))             -- drop1 パターン1 適用
= $Cons(2, $Cons(3, $Nil))                      -- force 適用
```

```
> drop2 (1, $Cons(1, $Cons(2, $Cons(3, $Nil))))
= $force (drop' (1, $Cons(1, $Cons(2, $Cons(3, $Nil))))) -- drop2 適用
= $force (drop' (0, $Cons(2, $Cons(3, $Nil))))          -- drop' パターン3 適用
= $force ($Cons(2, $Cons(3, $Nil)))                     -- drop' パターン1 適用
= $Cons(2, $Cons(3, $Nil))                              -- force 適用
```

長さ1のリストから3要素ドロップする例を考える

```
> drop1 (3, $Cons(1, $Nil))
= $force ($drop1 (2, $Nil)) -- drop1 パターン3 適用
= $force ($force ($Nil))    -- drop1 パターン2 適用
= $force ($Nil)             -- force 適用
= $Nil                      -- force 適用
```

```
> drop2 (3, $Cons(1, $Nil))
= $force (drop' (3, $Cons(1, $Nil))) -- drop2 適用
= $force (drop' (2, $Nil))           -- drop' パターン3 適用
= $force ($Nil)                      -- drop' パターン2 適用
= $Nil                               -- force 適用
```