


Homework 3

Submission is on this page (<https://canvas.umn.edu/courses/391238/assignments/3474029>).

In this assignment, you will be implementing three functions using calls to `List.fold_left` instead of recursion.

Submission: hw3_rec.ml, hw3.ml

- **hw3_rec.ml:** Should contain solutions to P1, P2, and P3. Use of recursion is optional (i.e. use of `rec` in function declarations). 90% of total score.
- **hw3.ml:** Should contain solutions to P1, P2, and P3 without the use of recursion by using calls to `List.fold_left`. 10% of total score. For this file, no points will be awarded if `rec` is detected. *Hint: it may help to write a helper accumulator function to use as an input to the `List.fold_left` call.*
- Submit both files at the gradescope link [here](https://www.gradescope.com/courses/612221)  (<https://www.gradescope.com/courses/612221>)

P1) `leftover_count : int list -> int -> int = <fun>`

Suppose we have a shuffled deck of numbered cards. Consider a card game where you select an initial number, and discard a batch of that size (or fewer, if not enough cards remain) from the top of the deck. The top card becomes the new number, and the process repeats. For example, if the top card is two, that card and the next card are discarded and the next card flipped over. The problem is to find the number of cards that still need to be discarded once the bottom of the deck is reached and there are no more cards to discard.

Function `leftover_count` takes in an integer list `lst` and an integer `x`, and outputs an integer. Input `lst` acts as the deck of cards, and `x` is the initial number, which can be thought of as the counter. A sketch of a possible implementation: each time the counter decrements, discard the first element in the list, so long as the list is non-empty. If the counter hits 0 and the list is non-empty, reset the counter with the value of the current element. If the list is empty, return the current value of the counter.

Assume the list contains only positive integers.

To illustrate, suppose `lst` = [1;4;2;10;1;4;2;10] and `x` = 2. At each step, decrement the counter and discard the head element in the list.

Counter = 2, lst = [1;4;2;10;1;4;2;10]

Counter = 1, lst = [4;2;10;1;4;2;10]

Counter = 0, lst = [2;10;1;4;2;10]; reset, Counter = value of current element = 2

Counter = 1, lst = [10;1;4;2;10]

Counter = 0, lst = [1;4;2;10]; reset, Counter = value of current element = 1

Counter = 0, lst = [4;2;10]; reset, Counter = value of current element = 4

Counter = 3, lst = [2;10]

Counter = 2, lst = [10]

Counter = 1, lst = []; empty list, return Counter.

So in this example, leftover_count [1;4;2;10;1;4;2;10] 2 = 1

Similarly,

leftover_count [1;4] 5 = 3

leftover_count [1;4] 2 = 0

leftover_count [1;4;2;10;3;5;7;11] 3 = 5

leftover_count [1;4;2;10;3;5;7;11] 5 = 2

leftover_count [1;4;2;10;3;5;7;11] 7 = 10

P2) last_card : int list -> int -> int = <fun>

Similar to P1, except the function should return the value of the last position (last selected card) where the counter reset. If the initial counter value does not point to a valid position in the list (i.e. initial value larger than the size of the list), return -1.

last_card [1;4] 5 = -1

last_card [1;4] 1 = 4

last_card [1;4;2;10;3;5;7;11] 1 = 5

last_card [1;4;2;10;3;5;7;11] 3 = 10

last_card [1;4;2;10;3;5;7;11] 7 = 11

P3) num_greater_than_sum : int list -> int = <fun>

Move along the list, keeping a sum total of all previously encountered elements. Return the total number of elements greater than the sum of previous elements.

To illustrate, let lst = [1;4;2;10]

[1;4;2;10]: Sum of previous elements = 0, current element = 1, add to count.

[1;4;2;10]: Sum of previous elements = 1, current element = 4, add to count.

[1;4;2;10]: Sum of previous elements = 5, current element = 2, count unchanged.

[1;4;2;**10**]: Sum of previous elements = 7, current element = 10, add to count.

Finished traversing list, return count.

So in this example, `num_greater_than_sum [1;4;2;10] = 3`

Similarly,

`num_greater_than_sum [] = 0`

`num_greater_than_sum [1;-1;2] = 2`

`num_greater_than_sum [1;4;2;10;3;-5;-7;11] = 4`