Q 1

Define `LENGTH` using `MAPCAR`, `LAMBDA`, `+` and `APPLY`.

Q 2

Define a recursive function `SUBS` that returns a sublist of a given list, with key parameters `start` and `end`. `(subs '(a b c d e) :start 1 :end 3)` should give `'(b c)`.

Q 3

`REDUCE` is a built-in that takes a two-parameter function and a list. It operates by first applying the function to the first two members, obtaining a result; then it applies the function to this result and the third member, obtaining another result; then it applies the function to the latest result and the fourth member, and so on. For instance `(reduce #'+ '(1 2 3 4))` will first add 1 and 2 obtaining 3, then add 3 and 3 obtaining 6, then add 6 and 4 finally obtaining and returning 10. Define your own `REVERSE` – the function that reverses a list, using `REDUCE`, `LAMBDA` and `CONS`.

Q 4

Define a recursive function `POS+` that takes a list and returns a version where each element is summed with its position – position count starts at 0. For instance `(pos+ '(7 5 1 4))` should give `(7 6 3 7)`.

Q 5

Define a function `PREC` that takes a symbol and a list, and returns all the members in the list that immediately precede the given symbol. For instance `(prec 'a '(a 1 a 2 b b 2 a a a))` should return `(1 2 A A)`. Define both a recursive and iterative version.

Q 6

The Fibonacci sequence is an infinite series of numbers starting with 0 and 1, where the rest of the numbers are computed by adding the previous two numbers in the series:

$$0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89 \ldots$$

for instance, we obtain 55 by adding 34 and 21; 89 by adding 55 and 34.

There is a one-to-one correspondence between the set of non-negative integers and the Fibonacci sequence, in the sense that every non-negative integer corresponds to one and only one Fibonacci number.

Integer	0	1	2	3	4	5	6	7	8	9	10	11…
Fibonacci Number	0	1	1	2	3	5	8	13	21	34	55	89…

Define `FIBO` that returns the corresponding Fibonacci number for a given non-negative integer.

Q 7

Given a list, remove all the elements that come between two occurrences of the symbol `T`. For instance, `(A B T C T T G)` should give `(A B T T T G)`, `(A B T C T G T G A T C)` should give `(A B T T T T C)`.

Q 8

Define a function `DEFRAG` that takes a list of zero or more `*`s and `+`s and rearranges them so that all the stars precede the pluses. For instance `(defrag '(+ + * + * +))` gives `(* * + + + +)`.

Q 9

Take an integer `n` and generate `n` many random tosses of a fair coin. Your output should be a list of `H`s and `T`s.

Q 10

Given a list of coin tosses, count the number of successes – let head be a successful, and tail be an unsuccessful toss.

Q 11

Given a list of coin tosses, calculate the proportion of Hs. For instance, a call like `(berno (toss-coin 100))` should give a number between 0 and 1. Remember that you can convert ratios to floating point numbers via `FLOAT`.

Q 12

Let us call "an experiment" a hundred successive tosses of a fair coin. You can represent an experiment as a hundred element list of Hs and Ts. Take an integer `n` and generate `n` many experiments, collecting them in a list.

Q 13

Given a list of values, construct a frequency table. For instance `(freq-table '(a b r a c a d a b r a))` should give `((A . 5) (R . 2) (B . 2) (D . 1) (C . 1))`, or `(freq-table '(1 0 1 1 1 0 0 1 0 0 1))` should give `((1 . 6) (0 . 5))`.

Q 14

Take an integer `n`, generate `n` many experiments, and tabulate the frequency of heads across the experiments. For instance a run for 100000 experiments – each of them 100 tosses of a fair coin – may give

```
((46 . 5799)
 (50 . 7853) (41 . 1593) (54 . 5907) (51 . 7716)
 (56 . 3949) (57 . 2993) (45 . 4934) (43 . 3026)
 (49 . 7761) (52 . 7227) (55 . 4884) (44 . 3853)
 (47 . 6638) (48 . 7379) (42 . 2245) (53 . 6866)
 (58 . 2193) (60 . 1093) (40 . 1063) (59 . 1537)
 (38 . 461) (61 . 693) (64 . 155) (71 . 1) (37 . 295)
 (35 . 85) (39 . 670) (33 . 26) (62 . 458) (68 . 10)
 (63 . 244) (36 . 156) (70 . 4) (65 . 77) (66 . 58)
 (34 . 51) (67 . 26) (32 . 12) (31 . 6) (72 . 1)
 (69 . 1) (73 . 1))
```

meaning, for instance, that in 6638 of 100000 experiments the number of heads was 47.

Q 15

The output for the previous exercise would look much informative if it is sorted. The builtin function `SORT` takes two arguments. One is the list (or sequence) to be sorted and the other is a two argument predicate (= a function that returns `T` or `NIL`) that compares its arguments. The predicate should be given in such a way that when it gives `T`, it would mean the first argument precedes the second. For instance, to sort a list of integers named `intlist` in ascending order (from smaller to larger), do `(sort intlist #'<)`. The predicate can be as complex as you like – lambda's are very helpful with sort.

Sort your freq-tables with respect to first and second components.