

Homework 3

1. Write a function in Lisp which behaves as CAR (without using CAR).
2. Write a function in Lisp which behaves as CDR (without using CDR).
3. Write a function in Lisp which behaves as CONS (without using CONS).
4. Write a function in Lisp which behaves as SECOND (without using SECOND).
5. Write a function in Lisp which behaves as THIRD (without using THIRD).
6. Write a function in Lisp which behaves as + (without using +).
7. Explore ¹ the theory of natural numbers using Lisp. We know,
 - 0 is natural number,
 - if x is natural number, then also $s(x)$ (the successor) is natural number, and
 - we define these in Lisp as a function `nat/1` which is true when its argument is a natural number:

```
0 is natural number
If x is natural number, then the successor of x
is natural number.
```

With 0 and s (the function for successor) you can define the sum $+$ of two natural numbers x, y :

$$\begin{aligned}0 + y &= y, \\ s(x) + y &= s(x + y).\end{aligned}$$

8. Write a function in Lisp which recognizes a palindrome².
9. Write a function in Lisp that returns the maximum element from a list of integers.
10. Write a function `shift_left (List1)` such that the returned list is List1 "as a rotational shift" with one element to the left. Example:

```
> (shift_left '(1 2 3 4 5))
(2 3 4 5 1)

> (shift_left '(2 3 4 5 1))
(3 4 5 1 2)
```

¹Define addition (already done as an example), multiplication, exponentiation, less than, less or equal than, divide, subtraction (attention, only the binary version matters), is divisible by

²A palindrome is a word, a phrase, a number (or any other sequence of objects) which has the property that if read from any direction is the same (you are allowed to use spaces between words). [source:wikipedia.org]

11. Write a function `shift_right (List1)` such that the returned list is List1 "as a rotational shift" with one element to the right. Example:

```
> (shift_right '(1 2 3 4 5))
(5 1 2 3 4)

> (shift_right '(5 1 2 3 4))
(4 5 1 2 3)
```
12. Write a tail recursive program for calculate the factorial of a natural number.
13. Write a program `delete_vowels(String)` in order to delete the vowels from a string.
14. Write a program `change_string` in order to change a string by transforming all the vowels into V, all the consonants into C and all the remaining characters into 0.
15. Write a program `sum_of` which returns the sum of the elements from a list Write also the tail recursive version. Example:

```
> (sum_of '(1 -3 2 0))
0
```
16. Write a program `square_sum` which returns the sum of squares of the elements from a list. Write also the tail recursive version. Example:

```
> (square_sum '(1 -3 2 0))
14
```
17. Write a program `arithmetic_average` which calculates the arithmetic average of the numers from a list. Write also the tail recursive version. Example:

```
> (arithmetic_average '(1 -10 2 9))
1
```
18. Define a binary relation `prefix/2` between lists and their prefixes. Suggestion: `()`, `(a)` and `(a b)` are the prefixes of the list `(a b)`.
19. Define a binary relation `suffix/2` between lists and all their suffixes. Suggestion: `()`, `(b)` and `(a b)` are the suffixes of the list `(a b)`.
20. Define a binary relation `sublist/2` between lists and their sublists.
21. Implement in Lisp the insertion sort algorithm for lists which contain integer numbers. This algorithm can be described as:

Given a list, delete the head, sort the tail of the list, then insert the head of the list into the sorted version of the tail of the list such that the result will be a sorted list.
22. Implement in Lisp the selection sort algorithm for lists which contain numbers. This algorithm can be described as:

Given a list, find the minimum element and place it on the first position, and repeat the process for the tail of the list.

23. Implement in Lisp the quick sort algorithm for lists which contain numbers. This algorithm can be described as:

Given a list, split in two sublists – one sublist contains all the elements smaller than one element of the list (pivot) and the other sublist contains all the elements greater than the pivot. Then sort the two sublists and concatenate them.

24. Implement in Lisp the merge sort algorithm for lists which contain numbers. This algorithm can be described as:

Given a list, split in two sublists which have (almost) the same length. Sort the two sublists and then combine them such that the result will be a sorted list.

25. Write a function `two_times_longer(L1, L2)` which returns true if the list L2 is two times longer than L1. You are not allowed to calculate the length of the lists.

26. Write a function `fib(N,F)` which is true if F is the Nth Fibonacci³ number. Calculate `(fib 5)`, `(fib 10)`, `(fib 50)`. Write also the tail recursive version.

27. Implement the extended euclidian algorithm⁴ in order to calculate the greatest common divisor of two integer numbers.

28. Write a function `without_doubles_1(my-list)` which returns a list without doubles. The elements have to be in the same order as in the input list and keep the last occurrence of the element which occurs two (or multiple) times.

Example:

```
> (without_doubles_1 '(1 2 3 4 5 6 4 4))
(1 2 3 5 6 4)
```

29. Write a function `without_doubles_2(my-list)` which returns a list without doubles. The elements have to be in the reverse order of the input list and keep the first occurrence of the element which occurs two (or multiple) times.

Exemplu:

```
> (without_doubles_2 '(1 2 3 4 5 6 4 4))
(6 5 4 3 2 1)
```

30. Write a function `delete_all_occ (Element,my-List)` which deletes all the occurrences of an element from a list.

Exemplu:

```
> (delete_all_occ a '(a b c a d a))
(b c d)

> (delete_all_occ a '(b c d))
(b c d)
```

³See http://en.wikipedia.org/wiki/Fibonacci_number

⁴See http://en.wikipedia.org/wiki/Extended_Euclidean_algorithm

31. Write a function `delete_first (Element,my-List)` which deletes the first occurrence of an element from a list.

```
> (delete_first a '(a b c a d a))  
(b c a d a)
```

```
> (delete_first a '(b c d))  
(b c d)
```

32. Write a function `count_occ(my-List)` which returns a list with pairs [element, number_of_occurrences_in_list], where 'element' is the element from the list my-List, and number_of_occurrences_in_list is an integer number corresponding to the number of occurrences of the element in the list. Such a pair has to appear for each of the elements from the list.

Exemplu:

```
> (count_occ '(a b a a b c))  
((c 1) (b 2) (a 3))
```

33. Write a function `element_position(element my-List)` which returns the position of an element in a list.

Exemplu:

```
> (element_position 2 '(1 2 3 4 5))  
1
```

```
>(element_position 2 '(1 2 3 2 4 5))  
1
```

34. Write a function `delete_nth(my-list, n)` which deletes each Nth element from a list.

Exemple:

```
> (delete_nth '(a b c d e f) 2)  
(a c e)
```

```
> (delete_nth '(a b c d e f) 1)  
nil
```

```
> (delete_nth '(a b c d e f) 0)  
nil
```

```
> (delete_nth '(a b c d e f) 10)  
(a b c d e f)
```

35. Write a function `even_num(my-list)` which returns the list of even numbers from a list. Do not use the predicate `EVENP`. Exemple:

```
> (even_num '(1 2 3 4 4 5 7 8))  
(2 4 4 8)
```

36. Write a function `odd_num(my-list)` which returns the list of odd numbers from a list. Do not use the predicate `ODDP`. Exemple:

```
> (odd_num '(1 2 3 4 4 5 7 8))
(1 3 5 7)
```

37. Write a function `module_of(my-list)` which returns the list containing the absolute value of the numbers from a list. Do not use the function `ABS`.
Exemple:

```
> (module_of '(1 -2 3 -4 4 5 -7 8))
(1 2 3 4 4 5 7 8)
```

38. Write a function `equal-length(list1, list2)` which returns `T` if the two lists have the same length and `NIL` otherwise.
39. Write a function `order(elem1, elem2, my-list)` which returns `T` if `elem1` occurs before `elem2` in the list and `NIL` otherwise.

Exemple:

```
> (order 1 2 '(1 2 3 4 4 5 7 8))
T
```

```
> (order 'a 'b '(d a c f g glg b))
T
```

```
> (order 'a 'b '(o y b l a c f g glg))
NIL
```

```
> (order 'a 'b '(d a c f g glg))
NIL
```

40. Write a function `equal_elem(list1, list2)` which returns `T` if `list1` and `list2` have the same elements.

Examples:

```
> (equal_elem '(1 2 3 4) '(1 2 3 4))
T
```

```
> (equal_elem '(a d c) '(a d c))
T
```

```
> (equal_elem '(a c d) '(a c))
NIL
```

```
> (equal_elem '(a c) '(a c d))
NIL
```

```
> (equal_elem '() '(1))
NIL
```

```
> (equal_elem '() '())
NIL
```