Tema 3

1. Scrieţi o funcţie în Lisp care să simuleze comportamentul lui CAR
2. Scrieţi o funcţie în Lisp care să simuleze comportamentul lui CDR
3. Scrieţi o funcţie în Lisp care să simuleze comportamentul lui CONS
4. Scrieţi o funcţie în Lisp care să simuleze comportamentul lui SECOND
5. Scrieţi o funcţie în Lisp care să simuleze comportamentul lui THIRD
6. Scrieţi o funcţie în Lisp care să simuleze comportamentul lui +
7. Exploraţi [1] teoria numerelor naturale folosind Lisp. Spre aducere aminte,
   - 0 este un număr natural,
   - dacă x este un număr natural, atunci si $s(x)$ (succesorul său) este, şi
   - acestea se definesc în Lisp ca o funcţie nat/1 care este adevărată când argumentul său este un număr natural:

     ```
     0 e numar natural
     Daca x e natural, atunci si succesorul lui x
     e numar natural.
     ```

   Cu 0 şi s, funcţia succesor, se poate defini suma + a două numere naturale x, y:

   $$0 + y = y,$$
   $$s(x) + y = s(x + y).$$

8. Scrieţi o funcţie care recunoaşte palindroame[2].
9. Să se scrie o funcţie pentru determinarea maximului unei liste de întregi.
10. Să se definească o funcţie shift_stg (List1) astfel ca lista returnată este List1 "cu un shift rotaţional" cu un element spre stânga. Exemplu:

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

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

11. Să se definească o funcţie shift_dr (List1) astfel ca lista returnată este List1 "cu un shift rotaţional" cu un element spre dreapta. Exemplu:

[1] Definiţi adunarea (deja făcută ca exemplu), înmulţirea, exponenţierea, mai mic, mai mic egal, divide, minus (atentie, numai varianta binara are sens), diviziunea

[2] Un palindrom este un cuvânt, frază, număr (sau orice altă secvenţă de obiecte) care are proprietatea că citit/ă (parcurs/ă) din orice direcţie arată la fel (ajustarea punctuaţiei şi spaţiilor dintre cuvinte este permisă). [sursa:wikipedia.org]

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

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

12. Scrieţi un program pentru calcului funcţiei factorial. Să fie eficient (final recursiv).

13. Scrieţi un program sterge_vocale(Sir) care şterge vocalele dintr-un şir de caractere.

14. Scrieţi un program schimba_sir care schimbă un şir de caractere prin transformarea tuturor vocalelor în reprezentarea lor cu literă mare, toate consoanele în reprezentarea lor cu literă mică şi toate celelalte caractere în 0.

15. Scrieţi un program suma care dintr-o listă de numere calculeaza suma acestora. Scrieţi şi varianta final recursivă. Exemplu:

```
> (suma '(1 -3 2 0))
0
```

16. Scrieţi un program suma_patrate care dintr-o listă de numere calculeaza suma pătratelor acestora. Scrieţi şi varianta final recursivă. Exemplu:

```
> (suma_patrate '(1 -3 2 0))
14
```

17. Scrieţi un program media_aritmetica care dintr-o listă de numere calculeaza media aritmetică a acestora. Scrieţi şi varianta final recursivă. Exemplu:

```
> (media_aritmetica '(1 -10 2 9))
1
```

18. Definiţi o relaţie binară prefix/2 între liste şi prefixele acestora. Sugestie: (), (a) şi (a b) sunt prefixele listei (a b).

19. Definiţi o relaţie binară sufix/2 între liste şi toate sufixele lor. Sugestie: (), (b) şi (a b) sunt sufixele listei (a b).

20. Definiţi o relaţie binară sublista/2 între liste si sublistele lor.

21. Implementaţi algoritmul de sortare prin inserţie liste de întregiîn Lisp– informal, acesta poate fi formulat astfel:

    Fiind dată o listă, se elimină capul ei, se sortează coada, iar apoi capul se introduce în lista sortată într-o poziţie ce păstrează lista sortată.

22. Implementaţi algoritmul de sortare prin selecţie pentru liste de întregi în Lisp – informal, acesta poate fi formulat astfel:

    Fiind dată o listă, se găseşte elementul minim, se plasează pe prima poziţie şi se repetă procesul pentru coada listei.

23. Implementaţi algoritmul de sortare rapidă pentru o listă de întregi în Lisp – informal, acesta poate fi formulat astfel:

    Fiind dată o listă, se împarte în două – o parte conţine elemente mai mici decât un element al listei (pivot) iar cealaltă conţine elementele mai mari. Cele două părţi se sortează iar variantele sortate se concatenează.

24. Implementaţi algoritmul de sortare prin intercalare pentru o listă de întregi în Lisp – informal, acesta poate fi formulat astfel:

    Fiind dată o listă, de împarte în două părţi de mărimi egale. Se sortează cele două părţi, iar rezultatele sortate se combină prin intercalare astfel ca ordinea elementelor să se păstreze.

25. Scrieţi o funcţie de_2_ori_mai_lung(L1, L2) care este satisfăcut dacă lista L2 este de două ori mai lungă decât L1. Calcularea lungimii listelor nu este permisă.

26. Scrieţi o funcţie fib(N,F) care este satisfă cută dacă F este al N-lea număr Fibonacci[3]. Să se calculeze (fib 5), (fib 10), (fib 50). Varianta final recursivă.

27. Implementaţi algoritmul euclidian extins[4] pentru calculul celui mai mare divizor comun a două numere întregi.

28. Scrieţi o funcţie fara_dubluri_1(lista) care este returnează lista fără dubluri. Elementele să fie în aceeaşi ordine în lista returnată ca şi în lista iniţială, păstrându-se ultima apariţie a elementelor duplicat.

    Exemplu:

```
> (fara_dubluri_1 '(1 2 3 4 5 6 4 4))
  (1 2 3 5 6 4)
```

29. Scrieţi o funcţie fara_dubluri_1(lista) care este returnează lista fără dubluri. Elementele să fie în ordine inversă în lista returnată ca şi în lista iniţială, păstrându-se prima apariţie a elementelor duplicat.

    Exemplu:

```
> (fara_dubluri_2 '(1 2 3 4 5 6 4 4))
(6 5 4 3 2 1)
```

30. Scrieţi o funcţie sterge_tot(Element,Lista) care şterge toate apariţiile lui Element în listă.

    Exemplu:

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

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

[3] Vezi http://en.wikipedia.org/wiki/Fibonacci_number
[4] Vezi http://en.wikipedia.org/wiki/Extended_Euclidean_algorithm

31. Scrieţi o funcţie sterge_primul(Element,Lista) care şterge prima apariţie a elementului din listă.

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

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

32. Scrieţi o funcţie numara_aparitii (Lista) care returnează listă perechi [element, numar_de_aparitii_in_lista], unde 'element' este un element din lista Lista, iar numar_de_aparitii_in_lista este un întreg ce dă numărul de apariţii ale elementului. O astfel de pereche trebuie să apară în rezultat pentru fiecare element al listei.

    Exemplu:

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

33. Scrieţi o funcţie pozitie_element (Lista) care returnează poziţia pe care se află un element într-o listă.

    Exemplu:

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

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

34. Scrieţi o funcţie sterge_nlea (lista , n) care şterge fiecare al N-lea element dintr-o listă.

    Exemple:

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

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

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

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

35. Scrieţi o funcţie pare(lista) care returnează lista cu numerele pare dintr-o listă. Este interzisă folosirea predicatului EVENP. Exemple:

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

36. Scrieţi o funcţie impare(lista) care returnează lista cu numerele impare dintr-o listă. Este interzisă folosirea predicatului ODDP. Exemple:

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

37. Scrieţi o funcţie modul(lista) care returnează lista cu numerele in valoare absolută dintr-o listă. Este interzisă folosirea funcţiei ABS. Exemple:

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

38. Scrieţi o funcţie lungimi-egale(lista1, lista2) care returnează T dacă cele două liste au aceeaşi lungime şi NIL altfel.

39. Scrieţi o funcţie ordine(elem1, elem2, lista) care returnează T dacă elem1 se află înaintea lui elem2 în lista şi NIL altfel.

    Exemple:

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

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

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

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

40. Scrieţi o funcţie elem_egale(lista1, lista2) care returnează T dacă lista1 şi lista2 au aceleaşi elemente.

    Exemple:

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

> (elem_egale '(a d c) '(a d c))
T

> (elem_egale '(a c d) '(a c))
NIL

> (elem_egale '(a c) '(a c d))
NIL

> (elem_egale '() '(1))
NIL

> (elem_egale '() '())
NIL
```