Exerciții laborator 4

1. Primul exercitiu presupune modificarea/adaugarea de instructiuni unui cod existent pentru a realiza anumite lucruri. In momentul actual programul aduna la elementul curent vecinul din dreapta sa (daca acesta exista).

```c
#include <stdio.h>

#define N 100

void sum_right_neighbour(int v[N], int n)
{
    int i;

    for (i = 0; i < n - 1; i++) {
        v[i] += v[i+1];
    }
}

void print_vector(int v[N], int n)
{
    int i;

    for (i = 0; i < n; i++) {
      printf("%d ", v[i]);
    }
    printf("\n");
}

int main(void)
{
    int v[N] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15,
16};

    print_vector(v, 16);
    sum_right_neighbour(v, 16);
    print_vector(v, 16);

    return 0;
}
```

Cerinte:

a. Sa se creeze o functie care sa adauge la fiecare element din vector vecinul din stanga sa (daca acesta exista), daca este nr par, altfel, daca este nr impar, va adauga partea intreaga a radicalului din valoarea vecinului din dreapta (daca acesta exista).

b. Sa se creeze o functie care sa construiasca un alt vector care va stoca in fiecare element vector[i] produsul tuturor elementelor care reprezinta puteri ale lui 2, mai putin elementul de pe pozitia i din vectorul initial (creati separat o functie care verifica ca un nr. este putere a lui 2).
c. Se citesc de la tastatura caractere (se va citi cate un caracter pe fiecare rand). In functie de valoarea acestuia se va realiza una din actiunile urmatoare in program:
    a. q - iesire din program
    b. m - eliminare element minim din vectorul initial
    c. M - eliminare element maxim din vectorul initial
    d. p - afisare vector

2. Pentru un vector de maxim 100 elemente citit de la tastatura (dimensiunea este si ea citita de la tastatura), afișați doar acele elemente din vector pentru care elementul este mai mare decât indicele său (păstrați ordinea din vectorul inițial). Considerați că primul element din vector se află la indicele 0.

```
Exemplu:
Input:      3 5 1 7 10 5 0
Output:     3 5 7 10
```

3. Pentru un vector de maxim 100 elemente citit de la tastatura (dimensiunea este si ea citita de la tastatura), afișați doar acele elemente din vector cu vecinii având valori mai mici decât el (păstrați ordinea din vectorul inițial). Prin vecini pentru elementul de pe poziția i se înțelege elementul de pe poziția i – 1, respectiv i + 1, cu două excepții: primul și ultimul element din vector (care au doar câte un vecin) – pentru aceste două elemente se va considera doar vecinul existent care trebuie să întrunească condiția pentu a afișa elementul. Atenție la ultimul element din vector (care nu este 0 sau valoarea negativă citită la final ci elementul anterior).

```
Exemplul 1:

Input:      3 5 1 7 10 5 0
Output:     5 10

Exemplul 2:

Input:      7 5 10 7 11 50 0
Output:     7 10 50

Exemplul 3:

Input:      7 0
Output:     7
```

4. Pentru un vector de maxim 100 elemente citit de la tastatura (dimensiunea este si ea citita de la tastatura), calculați media aritmetică a valorilor pare din vector. Dacă toate elementele vectorului sunt impare, afișați la consolă mesajul „Niciun element par in vector!”. Folosiți 2 variabile pentru calculul mediei aritmetice: sum pentru suma elementelor și counter pentru a număra câte elemente intră în calculul mediei. Veți afișa la consolă, rezultatul folosind instrucțiunea *printf(“%f\n”, sum / counter);*

   Studiați și explicați ce se afișează în fiecare din situațiile următoare pentru vectorul: 3 5 4 7 2 4 0

5. a) Pentru un vector de maxim 100 elemente intregi citite de la tastatura (dimensiunea este si ea citita de la tastatura), implementați algoritmul de sortare **Bubble Sort [1]** și afișați vectorul sortat descrescător.

   b) Folosind vectorul sortat cu Bubble Sort, implementați algoritmul de căutare binară iterativă (**Binary search [2]**)pentru a verifica dacă un număr întreg cu valoarea x citită de la tastatură se găsește în cadrul vectorului.

6. Se dă un vector de numere naturale (hardcodat sau citit de la tastatură, la alegerea voastră). Reordonați vectorul astfel încât fiecare element par se află înaintea tuturor elementelor impare: vectorul reordonat va conține toate elementele pare din vectorul initial (în ordinea din vector) urmate de toate elementele impare (în orice ordine). **Obligatoriu: NU trebuie să folosiți un vector adițional pentru implementare și trebuie să parcurgeți vectorul dat doar o singură dată pentru a genera rezultatul, iar apoi îl mai puteți parcurge o dată pentru afișare.**

   Exemplu input și output:

   ```
   Pentru vectorul:  3 5 1 7 10 5 0
   O soluție validă: 10 0 1 7 3 5 5
   O soluție invalidă: 0 10 1 7 3 5 5
   ```

7. Se dă un vector de numere naturale de lungime N în care se regăsesc toate numerele de la 1 la N - 1 (fiecare număr o singură dată, cu excepția unui singur număr care apare de 2 ori; elementele NU sunt sortate). Găsiți elementul duplicat eficient. Pentru implementare, puteți hardcoda un vector în cod, având caracteristicile descrise în enunț. (Hint: folosiți-vă de suma elementelor)

8. Se dă un vector de maxim 100 de numere întregi (dimensiunea este citita de la tastatura, la fel si elementele), care poate conține atât valori pozitive, cât și negative. Să se determine suma maximă a unei subsecvențe de elemente consecutive ale vectorului. (Hint: Algoritmul lui Kadane [3])

[1] https://en.wikipedia.org/wiki/Bubble_sort
[2] https://en.wikipedia.org/wiki/Binary_search
[3] https://en.wikipedia.org/wiki/Maximum_subarray_problem