FUNDAMENTELE PROGRAMĂRII

1. Se citește un vector cu n numere naturale. Să se afișeze câte elemente au valoare mai mare decât media aritmetică a elementelor vectorului dat.

Exemplu: pentru v = (10, 5, 8, 6, 3, 10, 8) => ma = 50 / 7 = 7.14 => există 4 valori cel puțin egale cu ma!

Rezolvare:

```
#include<stdio.h>
int main()
{
    int n, i, s, nrv, v[100];
    float ma;

    printf("Numarul de elemente din tablou: ");
    scanf("%d", &n);

    //citim elementele tabloului si calculam suma lor
    printf("Elementele tabloului:\n");
    s = 0;
    for(i = 0; i < n; i++)
    {
        printf("v[%d] = ", i);
        scanf("%d", &v[i]);
        s += v[i];
    }

    //calculam media aritmetica a valorilor din v
    ma=(float)s/n;

    //numaram cate elemente din v sunt mai marei decat ma
    nrv = 0;
    for(i = 0; i < n; i++)
        if(v[i] >= ma)
      nrv++;

    printf("\nMedia aritmetica a elementelor tabloului: %.2f", ma);
    printf("\nNumarul de valori din tablou care sunt mai mari decat
               media aritmetica: %d\n", nrv);

    return 0;
}
```

2. Se citește un vector format din n numere reale. Să se verifice dacă elementele sale sunt în ordine crescătoare sau nu.

Rezolvare:

Verificăm dacă `v[0]<=v[1]` și `v[1]<=v[2]` și ... și `v[n-2]<=v[n-1]`.

```
#include <stdio.h>

int main()
{
    int v[100], i, n;

    //citim elementele tabloului
    printf("Numarul de elemente din vector: ");
    scanf("%d", &n);

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

    //presupun ca v este sortat crescator
    for(i = 0; i < n-1; i++)
        //daca o pereche de elemente alaturate nu respecta
        //conditia v[i]<=v[i+1], atunci presupunerea ca v
        //ar fi sortat crescator devine falsa
        if(v[i] > v[i+1])
            break;

    if(i == n-1)
        printf("\nTabloul este sortat crescator!\n");
    else
        printf("\nTabloul nu este sortat crescator!\n");

    return 0;
}
```

Variantă:

```
#include <stdio.h>

int main()
{
    int n, i, v[101];

    printf("Numarul de elemente din vector: ");
    scanf("%d", &n);
```

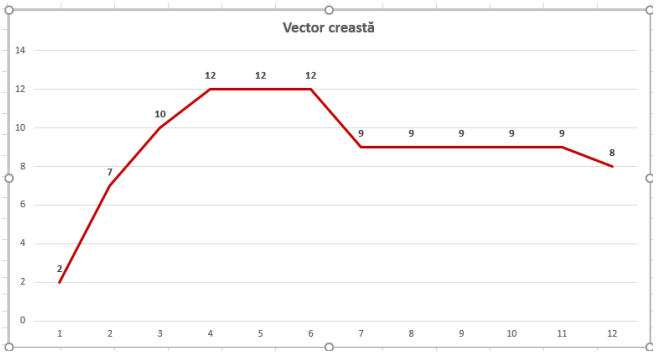
```
    for(i = 0; i < n; i++)
    {
        printf("v[%d] = ", i);
        scanf("%d", &v[i]);
    }

    i = 0;
    while(i < n-1 && v[i] <= v[i+1])
        i++;

    if(i == n-1)
        printf("Tabloul este sortat crescator!");
    else
        printf("Tabloul nu este sortat crescator!");

    return 0;
}
```

3. Un vector v format din n numere întregi se numește *vector creastă* dacă există un indice p astfel încât $v[0] \leq v[1] \leq \cdots \leq v[p]$ şi $v[p] \geq v[p+1] \geq \cdots \geq v[n-1]$. Scrieţi un program care citeşte un vector format din n numere întregi şi verifică dacă este vector creastă sau nu.

Commented [REB1]:

$v = (2, 7, 10, 12, 9, 8)$

$v = (2, 7, 10, 12, 12, 12, 9, 9, 9, 9, 9, 8)$



Observație: Indicele p este, de fapt, poziția pe care se află maximul din tabloul v!

Rezolvare:

Vom impune condiția suplimentară ca tabloul să nu fie o semi-creastă, adică să nu fie sortat crescător sau descrescător.

```c
#include <stdio.h>

int main()
{
    int v[100], i, n, p, sc, sd, vf;

    //citim elementele tabloului
    printf("Numarul de elemente din vector: ");
    scanf("%d", &n);

    //citim separat primul element din tablou
    printf("v[0] = ");
    scanf("%d", &v[0]);

    //presupunem ca tabloul este sortat crescator
    //sau descrescator
    sc = sd = 1;
    //citesc restul elementelor din tablou
    for(i = 1; i < n; i++)
    {
        printf("v[%d] = ", i);
        scanf("%d", &v[i]);
        //daca v[i] > v[i-1], atunci tabloul
        //nu este sortat descrescator
        if(v[i] > v[i-1])
            sd = 0;

        //daca v[i] < v[i-1], atunci tabloul
        //nu este sortat crescator
        if(v[i] < v[i-1])
            sc = 0;
    }

    //daca tabloul este sortat, atunci nu este creasta
    if(sc == 1 || sd == 1)
    {
        printf("\nTabloul nu este creasta!\n");
        return 0;
    }

    //"urc" spre creasta
    p = 0;
    while(p < n-1 && v[p] <= v[p+1])
        p++;

    //varful crestei este pozitia pana in care am urcat
    vf = p;

    //"cobor" de pe creasta
    while(p < n-1 && v[p] >= v[p+1])
        p++;
```

```
    if(p == n-1)
        printf("\nTabloul este creasta cu varful pe pozitia %d!\n", vf);
    else
        printf("\nTabloul nu este creasta!\n");

    return 0;
}
```

4. Operații cu mulțimi: intersecție, reuniune și diferență.

Rezolvare:
Vom presupune faptul că tablourile implicate sunt mulțimi, adică au elemente distincte.

A = (2, 1, 7, 10, 5, 21, 11) => m = 7 elemente
B = (11, 2, 35, 10, 18) => n = 5 elemente

- C = A ∩ B => mulțimea C poate să aibă cel mult min{m, n} elemente

  C = (2, 10, 11)

- D = A \ B => mulțimea D poate să aibă cel mult m elemente

  D = (1, 7, 5, 21)

- R = A U B => mulțimea R poate să aibă cel mult m+n elemente

  A = (2, 1, 7, 10, 5, 21, 11) => m = 7 elemente
  B = (11, 2, 35, 10, 18) => n = 5 elemente

  R = (2, 1, 7, 10, 5, 21, 11, 35, 18)

  R = A U (B\A)

```
#include <stdio.h>
```

```
int main()
{
    //cele doua multimi initiale
    int A[200], B[100];

    //intersectia si diferenta
    int C[100], D[200];

    //reuniunea
    int R[300];

    int i, j, m, n, p, q, k, g;

    //citim elementele multimii A
    printf("Numarul de elemente din multimea A: ");
    scanf("%d", &m);

    for(i = 0; i < m; i++)
    {
        printf("A[%d] = ", i);
        scanf("%d", &A[i]);
    }

    //citim elementele multimii B
    printf("\nNumarul de elemente din multimea B: ");
    scanf("%d", &n);

    for(i = 0; i < n; i++)
    {
        printf("B[%d] = ", i);
        scanf("%d", &B[i]);
    }

    //calculam intersectia si diferenta dintre A si B
    //p = numarul elementelor din intersectia C
    p = 0;
    //q = numarul elementelor din diferenta D = A\B
    q = 0;
    //cautam fiecare element din multimea A in multimea B
    for(i = 0; i < m; i++)
    {
        //presupunem ca A[i] nu se gaseste in multimea B
        g = 0;
        //compar A[i] cu fiecare element din B
        for(j = 0; j < n; j++)
            if(A[i] == B[j])
            {
                g = 1;
```

```
            break;
        }

    //daca A[i] se gaseste in multimea B, atunci
    //il adaug pe A[i] in intersectia C, respectiv
    //pe prima pozitie libera, adica pozitia p
    if(g == 1)
    {
        C[p] = A[i];
        p++;
    }
    //daca A[i] nu se gaseste in multimea B, atunci
    //il adaug pe A[i] in diferenta D, respectiv
    //pe prima pozitie libera, adica pozitia q
    else
    {
        D[q] = A[i];
        q++;
    }
}

if(p == 0)
    printf("\nIntersectia este vida (A si B sunt disjuncte)!\n");
else
{
    printf("\nIntersectia multimilor A si B este:\n");
    for(i = 0; i < p; i++)
        printf("%d ", C[i]);
}

if(q == 0)
    printf("\nMultimea A este inclusa in multimea B!\n");
else
{
    printf("\nDiferenta multimilor A si B este:\n");
    for(i = 0; i < q; i++) printf("%d ", D[i]);
}

//calculam reuniunea dintre A si B
//copiem toate elementele multimii A in reuniunea R
for(i = 0; i < m; i++)
    R[i] = A[i];

//k = numarul elementelor din reuniunea R
k = m;

//adaugam in reuniunea R elementele diferentei B\A
for(j = 0; j < n; j++)
```

```
    {
        g = 0;
        for(i = 0; i < m; i++)
            if(B[j] == A[i])
            {
                g = 1;
                break;
            }

        if(g == 0)
        {
            R[k] = B[j];
            k++;
        }
    }

    if(k == 0)
        printf("\nMultimile A si B sunt vide!\n");
    else
    {
        printf("\nReuniunea multimilor A si B este:\n");
        for(i = 0; i < k; i++) printf("%d ", R[i]);
    }


    return 0;
}
```