Laborator 2 – Structuri de date

2.1 Probleme rezolvate

P2.1 Să se realizeze un program ce permite stocarea numelui, prenumelui, salariului plătit pe oră, numărului de ore lucrate într-o lună şi a salariului total pentru fiecare din angajaţii unei companii (maxim 100 de angajați). Salariul total va fi calculat în funcţie de numărul de ore lucrate, astfel încât dacă numărul de ore lucrate pe lună depăşeşte valoarea 160, se adaugă un bonus de 50% la orele lucrate în plus. La final, programul va afişa pe ecran numele şi prenumele angajaţilor care au salariul lunar mai mare ca o valoare specificată.

Rezolvare:

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

int main()
{
 /* Declarare structura */
 struct date_angajat
 {
  char nume[256];
  char prenume[256];
  float salariu_pe_ora;
  int nr_ore;
  float salariu_final;
 };

 /* Declarare vector de structuri */
 struct date_angajat angajati_companie[100];

 /* Declarare alte variabile */
 int i, nr_angajati, ore_limita = 160;
 float salariu_minim;

 /* Citire numar angajati si verificare */
 do
 {
```

```
printf("Introduceti numarul de angajati ai companiei: ");
scanf("%d", &nr_angajati);

if ((nr_angajati<1)||(nr_angajati>100))
 printf("Numar invalid. Acesta trebuie sa fie intre 1 si 100.\n");
} while ((nr_angajati<1)||(nr_angajati>100));

/* Citire date angajati */
for (i=0; i<nr_angajati; i++)
{
 printf("\n#Citire date angajat %d#\n", i+1);
 printf("Nume: ");
 scanf("%s", angajati_companie[i].nume);
 printf("Prenume: ");
 scanf("%s", angajati_companie[i].prenume);
 printf("Salariu pe ora: ");
 scanf("%f", &angajati_companie[i].salariu_pe_ora);
 printf("Numar de ore lucrate: ");
 scanf("%d", &angajati_companie[i].nr_ore);
}

/* Citire salariu minim */
printf("\nIntroduceti salariul minim: ");
scanf("%f", &salariu_minim);

/* Calcul salariu final */
for (i=0; i<nr_angajati; i++)
{
 angajati_companie[i].salariu_final =
   angajati_companie[i].salariu_pe_ora *
   angajati_companie[i].nr_ore;

 if (angajati_companie[i].nr_ore>ore_limita)
 {
  angajati_companie[i].salariu_final +=
    angajati_companie[i].salariu_pe_ora *
    (angajati_companie[i].nr_ore-160) * 0.5;
  }
}

/* Afisare angajati cu salariul mai mare ca salariul minim */
```

```
 printf("\nAngajatii care au salariul mai mare de %.2f sunt:\n",
            salariu_minim);
 for (i=0; i<nr_angajati; i++)
 {
  if (angajati_companie[i].salariu_final>salariu_minim)
   printf("%s %s - %.2f\n", angajati_companie[i].nume,
               angajati_companie[i].prenume,
               angajati_companie[i].salariu_final);
 }

 printf("\n");
 system("PAUSE");
 return 0;
}
```

Programul va afişa pe ecran, după compilare şi execuţie următoarele:

```
Introduceti numarul de angajati ai companiei: 2

#Citire date angajat 1#
Nume: Popescu
Prenume: Marius
Salariu pe ora: 20
Numar de ore lucrate: 180

#Citire date angajat 2#
Nume: Matei
Prenume: Vlad
Salariu pe ora: 10
Numar de ore lucrate: 160

Introduceti salariul minim: 1600

Angajatii care au salariul mai mare de 1600.00 sunt:
Popescu Marius - 3800.00
```

Discuţie:

- Acest program pune în evidenţă folosirea structurilor de date pentru a stoca mai multe informaţii pentru angajaţii unei companii. Astfel, pentru fiecare angajat se stochează numele şi prenumele, salariul plătit pe oră, numărul de ore lucrate într-o lună şi salariul lunar;
- Structura ce conţine datele unui angajat este denumită `date_angajat` şi este formată din mai multe tipuri de date, corespunzătoare valorilor ce se doresc a fi stocate, şi anume: un şir de caractere de maxim 256 de elemente pentru nume şi prenume – `char nume[256], char`

`prenume[256]`, două numere reale pentru plata pe oră şi salariul final - `float salariu_pe_ora`, `float salariu_final` şi un număr întreg pentru numărul de ore lucrate într-o lună - `int nr_ore`;

- Pentru stocarea datelor a mai mulţi angajaţi, având în vedere că pentru fiecare persoană se stochează aceleaşi informaţii, se creează un vector de structuri de tipul `date_angajat`. Vectorul creat este denumit `angajati_companie` şi permite stocarea datelor pentru cel mult 100 de angajaţi (`struct date_angajat angajati_companie[100]`);
- Accesarea datelor pentru fiecare angajat se efectuează prin intermediul câmpurilor corespunzătoare fiecărui element din şirul de structuri. De exemplu, pentru a obţine numele celui de-al doilea angajat, se accesează elementul de indice 1 al şirului `angajati_companie`, şi în cadrul acestuia câmpul `nume` prin intermediul următoarei formulări: `angajati_companie[1].nume`. Pentru a înţelege mai bine principiul de folosire a câmpurilor din structuri, putem presupune că această expresie este de fapt o simplă variabilă, iar regulile de citire şi de afişare, precum şi efectuarea de operaţii sunt similare lucrului cu variabile;
- Calculul salariului final pentru angajatul `i` se face multiplicând numărul de ore lucrate lunar cu salariul pe oră, astfel:

  `angajati_companie[i].salariu_pe_ora*angajati_companie[i].nr_ore`.

  Dacă numărul de ore depăşeşte valoarea 160, pentru orele suplimentare se mai adaugă un bonus de 50%, adică `nr_ore - 160` este multiplicat cu salariul pe oră, iar rezultatul final este înmulţit cu 1,5 (100%+50%);
- Pentru a afişa angajaţii cu salariul mai mare decât o valoare specificată (`salariu_minim`), este necesară parcurgerea tuturor elementelor şirului `angajati_companie`, iar acele persoane care îndeplinesc condiţia (salariul final să fie mai mare ca valoarea specificată) sunt afişate pe ecran prin câmpul `nume`.

P2.2 Să se realizeze un program ce permite stocarea de date pentru N fişiere audio. Fiecare fişier va conţine următoarele informaţii: un vector de lungime 10 cu valori numere întregi cuprinse între 0 şi 255 (secvenţa audio), un şir de caractere de lungime 20 (numele fişierului) şi un vector de lungime 10 cu valori numere reale, cuprinse între 0 şi 1 (secvenţa audio normalizată). Programul va calcula şi afişa pe ecran, pentru fiecare fişier, numele şi secvenţa audio normalizată, calculată după următoarea formulă:

$$secventa_{norm(i)} = \frac{secventa(i) - minim}{maxim - minim}$$

unde *secventa(i)* este valoarea elementului de indice *i* în cadrul secvenţei originale, *maxim* şi *minim* reprezintă valoarea maximă, respectiv minimă a secvenţei originale, iar *secventa_norm(i)* reprezintă valoarea normalizată corespunzătoare elementului de indice *i* din secvenţa originală.

Rezolvare:

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

/* Declarare structura */
struct fisier_audio
{
 char nume[256];
 char secventa_audio[10];
 float secventa_audio_normalizata[10];
};

/* Declarare vector de structuri */
struct fisier_audio grup_fisiere[20];

/* Citirea datelor se realizeaza printr-o functie */
void citire_fisiere(struct fisier_audio *grup_fisiere,
                    int nr_fisiere)
{
 int i, j;
 for (i=0; i<nr_fisiere; i++)
 {
  printf("\nIntroduceti date fisier %d:\n", i+1);
  printf("Nume fisier: ");
  scanf("%s",(grup_fisiere+i)->nume);

  for (j=0; j<10; j++)
  {
   printf("Secventa audio originala (%d): ", j+1);
   scanf("%d",&(grup_fisiere+i)->secventa_audio[j]);
  }
 }
}

/* Normalizarea valorilor se face printr-o functie */
void normalizare_secvente(struct fisier_audio *grup_fisiere,
                          int nr_fisiere)
{
 int i, j, minim, maxim;
 for (i=0; i<nr_fisiere; i++)
 {
  /* Calcul minim si maxim */
  minim = (grup_fisiere+i)->secventa_audio[0];
  maxim = minim;
  for (j=1; j<10; j++)
  {
```

```
    if ((grup_fisiere+i)->secventa_audio[j] < minim)
     minim = (grup_fisiere+i)->secventa_audio[j];
    if ((grup_fisiere+i)->secventa_audio[j] > minim)
     maxim = (grup_fisiere+i)->secventa_audio[j];
   }

   /* Normalizare elemente */
   for (j=0; j<10; j++)
    (grup_fisiere+i)->secventa_audio_normalizata[j] =
         (float)((grup_fisiere+i)->secventa_audio[j]-minim)/
         (maxim-minim);

 }
}

/* Afisarea secventei normalizate */
void afisare_secventa(float secventa_normalizata[10])
{
 int j;

 for (j=0; j<10; j++)
  printf("%.2f\t", secventa_normalizata[j]);
}

int main()
{
 /* Declarare variabile */
 int i, nr_fisiere;

 /* Citire numar fisiere si verificare */
 do
 {
  printf("Introduceti numarul de fisiere audio: ");
  scanf("%d", &nr_fisiere);

  if ((nr_fisiere<1)||(nr_fisiere>20))
   printf("Numar invalid. Acesta trebuie sa fie intre 1 si 20.\n");
 } while ((nr_fisiere<1)||(nr_fisiere>20));

 /* Citire fisiere */
 citire_fisiere(grup_fisiere, nr_fisiere);

 /* Calcul secvente audio normalizate */
 normalizare_secvente(grup_fisiere, nr_fisiere);

 /* Afisare nume si secvente audio normalizate */
 for (i=0; i<nr_fisiere; i++)
```

```
 {
  printf("\nSecventa normalizata pentru fisierul %s este: ",
                 grup_fisiere[i].nume);
  afisare_secventa((grup_fisiere+i)->secventa_audio_normalizata);
 }

 printf("\n");
 system("PAUSE");
 return 0;
}
```

Programul va afişa pe ecran, după compilare şi execuţie:

```
Introduceti numarul de fisiere audio: 2

Introduceti date fisier 1:
Nume fisier: Fisier_1
Secventa audio originala (1): 0
Secventa audio originala (2): 10
Secventa audio originala (3): 20
Secventa audio originala (4): 30
Secventa audio originala (5): 40
Secventa audio originala (6): 50
Secventa audio originala (7): 60
Secventa audio originala (8): 70
Secventa audio originala (9): 80
Secventa audio originala (10): 90

Introduceti date fisier 2:
Nume fisier: Fisier_2
Secventa audio originala (1): 90
Secventa audio originala (2): 80
Secventa audio originala (3): 70
Secventa audio originala (4): 60
Secventa audio originala (5): 50
Secventa audio originala (6): 40
Secventa audio originala (7): 30
Secventa audio originala (8): 20
Secventa audio originala (9): 10
Secventa audio originala (10): 0

Secventa normalizata pentru fisierul Fisier_1 este: 0.00    0.11
0.22    0.33    0.44    0.56    0.67    0.78    0.89    1.00

Secventa normalizata pentru fisierul Fisier_2 este: 1.00    0.89
0.78    0.67    0.56    0.44    0.33    0.22    0.11    0.00
```

Discuţie:

- Programul pune în evidenţă utilizarea pointerilor pentru lucrul cu structuri. Structura creată în această problemă conţine un câmp pentru stocarea numelui unui fişier (`char nume[256]`), un câmp pentru stocarea valorilor ce constituie o secvenţă audio (`char secventa_audio[10]`), şi un câmp pentru stocarea valorilor *normalizate* ale secvenţei audio stocate anterior (`float secventa_audio_normalizata[10]`). Acestea din urmă sunt numere reale, fiind cuprinse în intervalul [0;1]. Prin normalizarea valorilor unei variabile înțelegem schimbarea plajei de valori a acesteia de la intervalul inițial, [min;max], la un interval dorit, de regulă ales să fie între [0; 1]. Normalizarea valorilor permite astfel compararea variabilelor cu plaje de valori diferite;
- Secvenţa de cod `struct fisier_audio grup_fisiere[20];` permite declararea unui vector de maxim 20 de elemente, elemente de tipul structurii create anterior. Fiind un vector, elementele sale pot fi accesate nu numai prin intermediul indicelui, ci şi prin intermediul pointerilor (se accesează un element-structură găsit la locaţia de memorie specificată). Astfel, în loc să se folosească o formulare de tip `grup_fisiere[i]`, se poate folosi formularea echivalentă `grup_fisiere+i`, adică adresa de memorie a elementului de indice *i* (adresa de memorie a primului element din vector decalată cu *i* unităţi). Având în vedere faptul că structura este reprezentată printr-un pointer, pentru a accesa un câmp al acesteia se folosește de această dată operatorul "->". De exemplu, pentru a accesa numele celui de-al treilea fişier, putem scrie: `(grup_fisiere+2)->nume`, iar pentru a accesa adresa de memorie a primei valori din cadrul secvenţei audio corespunzătoare celui de-al treilea fişier, putem scrie `&(grup_fisiere+2)->secventa_audio[0]`. Analizând în detaliu această secvenţă de cod, observăm următorul efect:
    1. `grup_fisiere+2`: accesează al treilea fişier (adresa de memorie a elementului-structură de indice 2);
    2. `(grup_fisiere+2)->secventa_audio[0]`: accesează prima valoare a câmpului `secventa_audio` din cadrul fişierului de indice 2;
    3. `&(grup_fisiere+2)->secventa_audio[0]`: reprezintă adresa de memorie a valorii menționate anterior.
- Funcţia `void citire_fisiere(struct fisier_audio *grup_fisiere, int nr_fisiere)` permite citirea numelor şi a secvenţelor audio pentru toate fişierele stocate în structură. Având în vedere că vectorul de structuri `grup_fisiere` este un parametru de intrare transmis prin adresă (pointer), conţinuturile elementelor acestuia pot fi modificate permanent, direct în interiorul funcţiei;
- Funcţia `void normalizare_secvente(struct fisier_audio *grup_fisiere, int nr_fisiere)` calculează secvenţele normalizate pentru toate fişierele (transmitere `grup_fisiere` prin adresă). Pentru normalizarea unei secvenţe, este

necesară calcularea în primă fază a minimului şi a maximului acesteia, după care se aplică formula de calcul din enunţul problemei;

- Funcţia `void afisare_secventa(float secventa_normalizata[10])` permite afişarea unei singure secvenţe normalizate, de aceea aceasta trebuie apelată în programul principal pentru fiecare fişier în parte: `afisare_secventa((grup_fisiere+i)->secventa_audio_normalizata);` S-a optat pentru acest mod de vizualizare pentru a exemplifica posibilitatea transmiterii prin adresă a unui câmp al structurii, şi nu a întregii structuri, ca la celelalte funcţii ilustrate anterior.

P2.3 Să se realizeze un program ce permite stocarea imaginilor unei secvenţe video. Fiecare imagine va conţine următoarele informaţii: o matrice de dimensiune *m* x *n* de valori pe 4 biţi (imagine), un şir de caractere de maxim 10 elemente (nume) şi histograma imaginii (numărul de apariţii ale fiecărei valori din matrice).

Rezolvare:

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

/* Declarare structura */
struct imagine
{
 char *matrice;
 char nume[10];
 char hist[16];
};

/* Declarare vector de structuri */
struct imagine *secventa_video;

/* Alocarea si citirea unei imagini cu dimensiunile acesteia */
char *citire_matrice(int *dim1, int *dim2)
{
 int i, j;
 char *M;

 /* Se citesc dimensiunile imaginii */
 if ((*dim1 == 0) && (*dim2 == 0))
 {
  printf("Introduceti numarul de linii: ");
  scanf("%d", dim1);
  printf("Introduceti numarul de coloane: ");
  scanf("%d", dim2);
 }
```

```
  /* Alocare dinamica memorie *dim1 x *dim2 x 1 byte */
  M = (char *)malloc((*dim1)*(*dim2)*sizeof(char));
  if (!M)
  {
   printf("\nMemoria nu a putut fi alocata!");
   exit(1);
  }

  printf("Citire valori matrice:\n");
  for (i=0; i<*dim1; i++)
   for (j=0; j<*dim2; j++)
   {
    printf("M[%d][%d] = ", i+1, j+1);
    scanf("%d", M+*dim2*i+j);
   }

  // Returnare pointer la memoria alocata
  return M;
}

/* Afisare imagine */
void afisare_matrice(char *M, int dim1, int dim2)
{
 int i,j;

 for (i=0; i<dim1; i++)
 {
  for (j=0; j<dim2; j++)
   printf("%3d", *(M+dim2*i+j));
  printf("\n");
 }
}

/* Calcul histograme imagini */
void calcul_histograme(struct imagine *secventa_video, int
                       nr_imagini, int dim1, int dim2)
{
 int k, i, j;
 char *matrice;

 for (k=0; k<nr_imagini; k++)
 {
  /* Preluare pointer la imaginea curenta */
  matrice = (secventa_video+k)->matrice;

  /* Initializare cu 0 */
  for (j=0; j<16; j++)
```

```
    (secventa_video+k)->hist[j] = 0;

   /* Incrementeaza valoarea de la pozitia indicata de
      valoarea elementului matricei */

   for (i=0; i<dim1; i++)
    for (j=0; j<dim2; j++)
     (secventa_video+k)->hist[*(matrice+i*dim2+j)]++;

 }
}

/* Eliberare memorie */
void eliberare_structura(struct imagine *secventa_video)
{
 // verificare daca structura a fost alocata
 if (secventa_video!=NULL)
  free(secventa_video);
}

int main()
{
 /* Declarare variabile */
 int nr_imagini, dim1=0, dim2=0, k;

 /* Citire numar de imagini */
 printf("\nIntroduceti numarul de imagini din secventa video: ");
 scanf("%d", &nr_imagini);

 /* Alocare dinamica a spatiului de memorie pentru structura */
 secventa_video = (struct imagine *)malloc(nr_imagini*
                      sizeof(struct imagine));

 /* Citire nume si matrice imagini */
 for (k=0; k<nr_imagini; k++)
 {
  printf("\nIntroduceti date imagine %d:\n", k+1);
  printf("Nume imagine: ");
  scanf("%s", (secventa_video+k)->nume);
  printf("Matrice:\n");
  (secventa_video+k)->matrice = citire_matrice(&dim1,&dim2);
 }

 /* Afisare matrice */
 for (k=0; k<nr_imagini; k++)
 {
  printf("\nMatricea %d este:\n", k+1);
```

```
  afisare_matrice((secventa_video+k)->matrice, dim1, dim2);
 }

 /* Calcul histograme */
 calcul_histograme(secventa_video, nr_imagini, dim1, dim2);

 /* Afisare histograme */
 for (k=0; k<nr_imagini; k++)
 {
  printf("\nHistograma pentru imaginea %s este:\n",
          (secventa_video+k)->nume);
  afisare_matrice((secventa_video+k)->hist, 1, 16);
 }

 /* Eliberare memorie */
 eliberare_structura(secventa_video);

 printf("\n");
 system("PAUSE");
 return 0;
}
```

Programul va afişa pe ecran, după compilare şi execuţie:

```
Introduceti numarul de imagini din secventa video: 2

Introduceti date imagine 1:
Nume imagine: Imagine_1
Matrice:
Introduceti numarul de linii: 2
Introduceti numarul de coloane: 3
Citire valori matrice:
M[1][1] = 0
M[1][2] = 0
M[1][3] = 4
M[2][1] = 5
M[2][2] = 15
M[2][3] = 15

Introduceti date imagine 2:
Nume imagine: Imagine_2
Matrice:
Citire valori matrice:
M[1][1] = 8
M[1][2] = 8
M[1][3] = 8
M[2][1] = 8
```

```
M[2][2] = 10
M[2][3] = 1

Matricea 1 este:
  0  0  4
  5 15 15

Matricea 2 este:
  8  8  8
  8 10  1

Histograma pentru imaginea Imagine_1 este:
  2  0  0  0  1  1  0  0  0  0  0  0  0  0  0  2

Histograma pentru imaginea Imagine_2 este:
  0  1  0  0  0  0  0  0  4  0  1  0  0  0  0  0
```

Discuţie:

- Acest program pune în evidenţă alocarea dinamică în lucrul cu structuri. Aşa cum în cazul unui vector/matrice de valori se poate aloca dinamic memoria, acelaşi lucru se poate face şi cu vectori/matrice de structuri. Mai mult, elementele unui vector de structuri pot conţine câmpuri (vectori/matrice) alocate dinamic;
- Pointerul `*secventa_video` (`struct imagine *secventa_video`) va stoca adresa de memorie a primului element de tip structură `imagine`. Acest element este o structură ce conţine o matrice, un nume şi un vector de valori, reprezentând histograma matricei;
- Alocarea dinamică a spaţiului de memorie pentru vectorul `*secventa_video` se realizează cu ajutorul funcţiei `malloc` astfel:

  ```
  (struct imagine *)malloc(nr_imagini*sizeof(struct imagine));
  ```

  Se alocă astfel în total un număr de octeţi dat de numărul de imagini multiplicat cu necesarul de memorie pentru stocarea unei structuri de tip `imagine`, calculat cu ajutorul funcţiei `sizeof()`;
- Funcţia `char *citire_matrice(int *dim1, int *dim2)` alocă spaţiu de memorie necesar stocării unei matrice de dimensiuni impuse de utilizator, şi returnează adresa de memorie a primului element al acestei matrice, care se asociază în programul principal câmpului `matrice`: `(secventa_video+k)->matrice = citire_matrice(&dim1, &dim2);` Alocarea dinamică de memorie se face folosind funcţia `malloc`. Funcţia `citire_matrice` mai permite şi citirea de la tastatură, o singură dată, a dimensiunilor matricei, dimensiuni ce sunt vizibile şi în cadrul programului principal, deoarece sunt transmise prin adresă;
- În cadrul funcţiei `void calcul_histograme(struct imagine *secventa_video, int nr_imagini, int dim1, int dim2)` se calculează histograma corespunzătoare fiecărei matrice a unei imagini. Histograma unei matrice este un vector de dimensiune numărul de valori posibile pe care le ia matricea și care în fiecare indice

stochează numărul de apariții al valorii indicelui în matrice (frecvență de apariție). Inițial, toate frecvențele de apariţie se initializează cu 0, după care se parcurge fiecare element al matricei, iar indicele vectorului-histogramă dat de valoarea acestui element se incrementează cu unu, astfel:

```
(secventa_video+k)->hist[*(matrice+i*dim2+j)]++;
```

De exemplu, dacă elementul de pe prima linie şi prima coloană a matricei are valoarea 3, atunci elementul de indice 2 (al treilea) al histogramei devine 1. Dacă mai există un alt element din cadrul matricei cu valoarea 3, acelaşi element al histogramei devine 2, și așa mai departe. Codul de mai sus se traduce pe etape prin:

1. `secventa_video+k`: accesează imaginea de indice *k* (adresa de memorie a elementului-structură de indice *k* al vectorului);
2. `(secventa_video+k)->hist`: accesează vectorul histogramă al imaginii de indice k (adresa de memorie a primului element al vectorului). Pentru simplificarea explicațiilor următoare, notăm acest vector cu *H*;
3. `*(matrice+i*dim2+j)`: accesează conţinutul de la adresa de memorie a elementului de pe linia de indice *i* şi coloana de indice *j* a matricei imaginii (echivalent cu valoarea acestui element). Pentru simplificarea explicațiilor următoare, notăm această valoare cu *val*;
4. `(secventa_video+k)->hist[*(matrice+i*dim2+j)]`: accesează valoarea elementului de indice *val* al vectorului *H*, și anume *H*[*val*];
5. `(secventa_video+k)->hist[*(matrice+i*dim2+j)]++`: incrementează valoarea elementului *H*[*val*];

- Funcţia `void eliberare_structura(struct imagine *secventa_video)` eliberează spaţiul de memorie alocat dinamic pentru vectorul de elemente-structuri `secventa_video`. Funcţia primeşte ca parametru de intrare adresa primului element al acestui vector.

2.2 Probleme propuse

1. Să se modifice problema rezolvată nr. 2.2, astfel încât să stocheze şi să calculeze, pentru fiecare fişier audio, și histograma normalizată, conform formulei:

$$hist\_norm(i) = \frac{\text{maxim} - hist(i)}{\text{maxim} - \text{minim}}$$

unde *hist(i)* este frecvenţa de apariţie a elementului de indice *i* în vector (secvenţa audio), *maxim* şi *minim* reprezintă frecvenţa maximă, respectiv minimă a vectorului.