Universitatea "**Politehnica**" din Bucureşti
Facultatea de Electronică, Telecomunicaţii şi Tehnologia Informaţiei

Programarea Calculatoarelor (limbajul C)

Curs 6 – Tipuri de date compuse

Prof. Bogdan IONESCU

2016-2017

Cuprins

6.1. Lucrul cu tablouri

Tablouri de date

> Un tablou este o **structură omogenă** formată dintr-o mulţime finită de elemente de acelaşi tip, denumit şi tip de bază.

Exemplu:

$$\begin{bmatrix} 3.4 & 3 & 1.5 \\ 2 & \boxed{2.2} & 1.9 \\ 3 & 2.4 & 8 \end{bmatrix}$$

informaţia este stocată sub formă matriceală

sursa principală de informaţie sunt cifrele,

sursa secundară de informaţie este dată de relaţiile de vecinătate.

2.2 are ca vecini la: N - **3**, S – **2.4**, V – **2** şi E – **1.9**.

Tablouri de date (continuare)

> Informaţia de vecinătate este esenţială şi constituie motivul folosirii tablourilor şi nu a unei liste de valori.

Exemplu: dispunem de 14 valori de **0**, 26 valori de **1** şi 9 valori de **0.5**.

- *valori independente în memorie*

0 1 0 1 0 0.5 0 0.5 0 1 0 0.5 1 *etc.*

- *valori cu vecinătate pe o direcţie*

1 1 1 1 1 1 1 1 0 0 0 0 1 1 1 0 0 0 0 1 1 1 0 0 0 0 etc.

Tablouri de date (continuare)

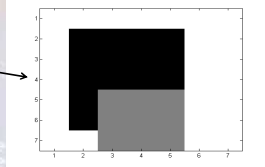
Exemplu: dispunem de 14 valori de **0**, 26 valori de **1** şi 9 valori de **0.5** (continuare)

- *valori cu vecinătate pe două direcţii*

1	1	1	1	1	1	1
1	0	0	0	0	1	1
1	0	0	0	0	1	1
1	0	0	0	0	1	1
1	0	0.5	0.5	0.5	1	1
1	0	0.5	0.5	0.5	1	1
1	1	0.5	0.5	0.5	1	1

datele încep să capete sens datorită informaţiilor suplimentare.

imagine cu două obiecte



Tablouri de date (continuare)

În continuare vom discuta despre următoarele tipuri de tablouri de date:

- **vectori de date**
- **matrice bidimensionale**
- **matrice n-dimensionale (n>2)**
- **şiruri de caractere**

Observaţie: însăşi modul de reprezentare al datelor în memorie este unul matriceal, astfel că lucrul cu tablouri de date (definire, manevrare) este eficient şi totodată favorizat.

Tablouri de date (continuare)

- **Vectori**

> Un vector reprezintă o colecţie de date de acelaşi tip reprezentate sub forma unei linii sau coloane.

Exemplu: [1 3 9 7] (indici 0 1 2 3) *sau* (indici 0, 1, 2, 3)

0	1
1	3
2	9
3	7

> Datele sunt identificate prin poziţia acestora în vector ce este dată de **indice.**

> În limbajul C vom folosi primul tip de reprezentare a vectorilor şi anume sub forma unei **linii.**

Atenţie: *indicele primului element este întotdeauna 0.*

Tablouri de date (continuare)

- **Vectori (continuare)**

> Modul de definire a unui vector în limbajul C:

<tip date> <nume_vector>[<număr elem.>];

- tipul datelor conţinute de vector: *ex.*: int, char, float
- numele variabilei de tip vector (identificator)
- numărul **maxim** de elemente ale vectorului

Exemple:

```
int x[100];
float y[50];
```

-variabila **x** este un vector cu 100 de valori *întregi*,
-variabila **y** este un vector cu 50 de valori *reale*.

Tablouri de date (continuare)

- **Vectori (continuare)**

> *Modul de alocare al memoriei unui vector:*

```
int x[4];
```

→ această modalitate de alocare a memoriei este de **tip static**, sistemul de calcul alocând în memorie numărul specificat de valori, indiferent dacă acestea vor fi folosite sau nu.

> Astfel, se alocă 4 locaţii de memorie consecutive de tip **int,** ce vor corespunde valorilor vectorului x:

32 biţi	32 biţi	32 biţi	32 biţi
adr. N	*adr. N+1*	*adr. N+2*	*adr. N+3*

> Fiecare locaţie este identificată prin adresa sa în memorie (adresele sunt consecutive).

Tablouri de date (continuare)

- **Vectori (continuare)**

> *Modul de accesare al valorilor vectorului din memorie:*

indici: 0	1	2	3
123	324	456	100
adr. N	*adr. N+1*	*adr. N+2*	*adr. N+3*

```
printf("%d",x);
```

x stochează **adresa** din memorie a primului element din vector (*N*),

```
printf("%d",x[2]);
```

x[2] reprezintă **valoarea** din a treia locaţie de memorie a vectorului x (456),

```
printf("%d",&x[3]);
```

operatorul **&** returnează **adresa** din memorie a variabilei ce il succede (N+3)

Tablouri de date (continuare)

- **Vectori (continuare)**

> *Exemplu citire vector:*

```
int v[10], i;
for (i=0; i<3; i++)
{
  printf("v[%d]=",i);
  scanf("%d",&v[i]);
}
```

Atenţie: în scanf se menţiona adresa la care va fi stocată valoarea citită, în acest caz &v[i].

Execuţie:

```
>v[0]=12 (enter)
>v[1]=45 (enter)
>v[2]=14 (enter)
```

> ***Observaţie***: nu este obligatoriu să folosesc toate valorile alocate, în acest caz 10, pot folosi un număr mai mic, dar nu un număr mai mare → **eroare**.

Rezultat: v[0]=12, v[1]=45, v[2]=14, v[3]=?, …, v[9]=?, ~~v[10]~~ …

Tablouri de date (continuare)

- **Vectori (continuare)**

Exemplu:

```
int v[200], dim, i, max;
printf("dimensiune vector=");
scanf("%d",&dim);
for (i=0; i<dim; i++)
{
  printf("v[%d]=",i);
  scanf("%d",&v[i]);
}
max=v[0];
for (i=1; i<dim; i++)
 if (v[i]>max)
  max=v[i];
printf("max=%d",max);
```

Ce face acest program ???

alocă un vector de maxim 200 de valori int, şi citeşte dimensiunea dorită (<200).

citeşte elementele vectorului.

maxim este primul element.

parcurge vectorul începând cu al doilea element şi determină dacă acesta este maxim.

Tablouri de date (continuare)

- **Vectori (continuare)**

P *Enunţ*: să se calculeze suma a doi vectori de numere reale. Vectorii au acelaşi număr de elemente (<100). Suma=v1[0]+v2[0]+ … + v1[N]+v2[N]

Variabile de intrare/lucru:

float v1[100], v2[100];
int dim;
int i;

Variabile de ieşire:

float suma;

Structură program:

- se citeşte dimensiunea dim,
- se citesc valorile vectorilor v1 şi v2,
- se parcurg vectorii şi se calculează suma valorilor.

Tablouri de date (continuare)

- **Vectori (continuare)**

```
float v1[100], v2[100], suma;
int i, dim;

printf("dim=");
scanf("%d", &dim);

for (i=0; i<dim; i++)        // citim ambii vectori într-un singur for
{
  printf("v1[%d], v2[%d]:", i, i);
  scanf("%f %f", &v1[i],&v2[i]);
}
suma=v1[0]+v2[0];        // iniţializăm suma cu primele elemente

for (i=1; i<dim; i++)
 suma+=v1[i]+v2[i];

printf("suma este:%.2f", suma);
```


Tablouri de date (continuare)

- **Vectori (continuare)**

P *Enunţ*: să se concateneze doi vectori de numere întregi de dimensiuni N şi respectiv M. Vectorul obţinut va fi stocat separat (N,M<100).

Variabile de intrare/lucru:

int v1[100], v2[100];
int N,M;
int i;

Variabile de ieşire:

int v[200];

Structură program:

- se citesc dimensiunile N şi M,
- se citesc valorile vectorilor v1 şi v2,
- se parcurge vectorul v1 → stocăm valori în v
- se parcurge vectorul v2 → stocăm valori în v.

Tablouri de date (continuare)

- **Vectori (continuare)**

```
int v1[100], v2[100], v[200];
int N, M, i;
printf("Introduceti N si M:");
scanf("%d %d", &N, &M);
for (i=0; i<N; i++)    // citim v1 şi profităm de for şi stocăm valori în v
{
  printf("v1[%d]=", i); scanf("%d", &v1[i]);
  v[i]=v1[i];
}
for (i=0; i<M; i++)   // citim v2 şi profităm de for şi stocăm valori în v
 {
  printf("v2[%d]=", i); scanf("%d", &v2[i]);
  v[N+i]=v2[i];
}
for (i=0; i<(N+M); i++)
  printf("%d", v[i]);
```


Tablouri de date (continuare)

- **Matrice 2D**

> O matrice 2D reprezintă o colecţie de date de acelaşi tip ce este structurată pe linii şi coloane (sub formă de tablou)

Exemplu:

$$\begin{array}{c|ccc} & 0 & 1 & 2 \\ \hline 0 & 2 & 9 & 1 \\ 1 & 1 & 7 & 2 \\ 2 & 4 & 5 & 3 \end{array}$$

linie

coloană

> Datele sunt identificate prin **indicele liniei** şi respectiv **indicele coloanei**.

Observaţie: vectorul este un caz particular de matrice cu o singură linie.

Tablouri de date (continuare)

- **Matrice 2D (continuare)**

> Modul de definire a unei matrice în limbajul C:

```
<tip date>  <nume_matrice>[<nr.linii>][<nr.coloane>];
```

- tipul datelor conţinute de matrice *ex.*: int, char, float
- numele variabilei de tip matrice (identificator)
- numărul **maxim** de linii şi coloane ale matricei

Exemple:

```
int x[10][10];
float y[5][10];
```

- variabila **x** este o matrice ce conţine 100 de valori *întregi*, pe 10 linii şi 10 coloane
- variabila **y** este o matrice ce conţine 50 de valori *reale*, pe 5 linii şi 10 coloane.

Tablouri de date (continuare)

- **Matrice 2D (continuare)**

> *Modul de alocare al memoriei unei matrice:*

```
int m[2][4];
```

→ alocarea este tot de **tip static**, sistemul de calcul alocând în memorie numărul specificat de valori.

> Astfel, se alocă 2x4 locaţii de memorie consecutive de tip **int**, ce vor corespunde liniilor şi coloanelor matricei **m**:

32 biţi	32 biţi	32 biţi	32 biţi
adr. N	*adr. N+1*	*adr. N+2*	*adr. N+3*
32 biţi	32 biţi	32 biţi	32 biţi
adr. N+4	*adr. N+5*	*adr. N+6*	*adr. N+7*

> *Observaţie:* adresele sunt de regulă consecutive.

Tablouri de date (continuare)

- **Matrice 2D (continuare)**

> *Modul de accesare al valorilor matricei din memorie:*

indici:	0	1	2	3
0	123	324	456	100
	adr. N	*adr. N+1*	*adr. N+2*	*adr. N+3*
1	34	5	258	199
	adr. N+4	*adr. N+5*	*adr. N+6*	*adr. N+7*

```
printf("%d",m);
```

m stochează **adresa** din memorie a primului element din matricea **m** (*N*),

```
printf("%d",m[1]);
```

m[1] reprezintă **adresa** liniei de indice 1, şi anume a primului element al acesteia (N+4),

Tablouri de date (continuare)

- **Matrice 2D (continuare)**

> *Modul de accesare al valorilor matricei din memorie (cont.):*

indici:	0	1	2	3
0	123	324	456	100
	adr. N	*adr. N+1*	*adr. N+2*	*adr. N+3*
1	34	5	258	199
	adr. N+4	*adr. N+5*	*adr. N+6*	*adr. N+7*

Observaţie: practic o matrice este o colecţie de vectori linie, în cazul de faţă avem doi vectori de adrese N şi N+4.

```
printf("%d",m[1][3]);
```

m[1][3] reprezintă **valoarea** de pe linia de indice 1, şi coloana de indice 3, a matricei **m** (199),

Tablouri de date (continuare)

- **Matrice 2D (continuare)**

> *Modul de accesare al valorilor matricei din memorie (cont.):*

indici:	0	1	2	3
0	123	324	456	100
	adr. N	*adr. N+1*	*adr. N+2*	*adr. N+3*
1	34	5	258	199
	adr. N+4	*adr. N+5*	*adr. N+6*	*adr. N+7*

```
printf("%d",&m[1][3]);
```

&m[1][3] reprezintă **adresa** de memorie a elementului de pe linia de indice 1 şi coloana de indice 3

Tablouri de date (continuare)

- **Matrice 2D (continuare)**

> *Exemplu citire matrice:*

```
int m[10][10], i, j;
for (i=0; i<2; i++)
 for (j=0; j<2; j++)
 {
  printf("v[%d][%d]=", i, j);
  scanf("%d",&m[i][j]);
 }
```

Atenţie: în scanf se menţiona adresa la care va fi stocată valoarea citită, deci &m[i][j].

Execuţie:

```
>v[0][0]=12 (enter)
>v[0][1]=45 (enter)
>v[1][0]=14 (enter)
>v[1][1]=42 (enter)
```

> ***Observaţie***: ca şi în cazul vectorilor, nu este obligatoriu să folosesc toate locaţiile alocate, pot folosi un număr mai mic de linii sau coloane, dar nu mai mare → **eroare**.

Tablouri de date (continuare)

- **Matrice 2D (continuare)**

Exemplu:

```
int m[5][10], Nl, Nc;
int i, j, prod=1;
printf("Nl si Nc:");
scanf("%d %d",&Nl, &Nc);
for (i=0; i<Nc; i++)
  for (j=0; j<Nl; j++)
  {
    printf("m[%d][%d]=", i, j);
    scanf("%d",&m[i][j]);
    prod*=m[i][j];
  }
printf("Prod=%d",prod);
```

Ce face acest program ???

alocă o matrice de maxim 5x10 de valori int, şi citeşte nr. linii (<5) şi nr. coloane (<10).

citeşte elementele matricei.

calculează produsul elementelor matricei v.

Este o greşeală importantă în program!

Tablouri de date (continuare)

- **Matrice 2D (continuare)**

P *Enunţ*: să parcurgă elementele de sub diagonala principală (inclusiv) ale unei matrice pătratice de numere reale (<20x20).

$$\begin{bmatrix} 2 & 9 & 1 \\ 1 & 7 & 2 \\ 4 & 5 & 3 \end{bmatrix}$$

Variabile de intrare/lucru:

float m[20][20];
int N;
int i, j;

Variabile de ieşire:

valori matrice,

Structură program:

- se citeşte dimensiunea N,
- se citesc valorile matricei m,
- se parcurg elementele de sub diagonala principală (i=j)

Tablouri de date (continuare)

- **Matrice 2D (continuare)**

```
float m[20][20];
int i, j, N;
printf("N="); scanf("%d", &N);
for (i=0; i<N; i++)        // citim matricea
  for (j=0; j<N; j++)
  {
    printf("m[%d][%d]:", i, j);
    scanf("%f", &m[i][j]);
  }
for (i=0; i<N; i++)        // parcurgem elementele de sub diagonală
{
  for (j=0; j<=i; j++)
    printf("%.2f ", m[i][j]);
  printf("\n");
}
```

i=0,…,2

$$\begin{bmatrix} 2 & 9 & 1 \\ 1 & 7 & 2 \\ 4 & 5 & 3 \end{bmatrix}$$

j=0,…,i

Tablouri de date (continuare)

- **Matrice 2D (continuare)**

P *Enunţ*: să se realizeze produsul a două matrice de numere întregi de dimensiuni diferite, MxN şi NxL.

Variabile de intrare/lucru:

int A[50][50], B[50][50];
int M, N, L;
int i, x, y;

Variabile de ieşire:

int C[50][50],

Problemă:

trebuie determinat **modul general de calcul** al unui element din matricea rezultantă,

Tablouri de date (continuare)

- **Matrice 2D (continuare)**

> Produsul a două matrice:

$$\underset{M \times N}{A} \times \underset{N \times L}{B} = \underset{M \times L}{C}$$

$$\begin{array}{c} N \\ M \end{array}\begin{bmatrix} 2_{0,0} & 9_{0,1} & 1_{0,2} \\ 1_{1,0} & 7_{1,1} & 2_{1,2} \\ 4_{2,0} & 5_{2,1} & 3_{2,2} \end{bmatrix} \times \begin{array}{c} L \\ N \end{array}\begin{bmatrix} 4_{0,0} & 5_{0,1} \\ 2_{1,0} & 2_{1,1} \\ 1_{2,0} & 8_{2,1} \end{bmatrix} = \begin{array}{c} L \\ M \end{array}\begin{bmatrix} \_{0,0} & \_{0,1} \\ \_{1,0} & \_{1,1} \\ \_{2,0} & \_{2,1} \end{bmatrix}$$

C[1][0]= A[1][0]*B[0][0] + A[1][1]*B[1][0] + A[1][2]*B[2][0]

>*Formula magică este:*

$$C[x][y] = \sum_{i=0}^{N-1} A[x][i] * B[i][y], x = 0,...,M-1; y = 0,...,L-1$$

Tablouri de date (continuare)

- **Matrice 2D (continuare)**

```
int A[50][50], B[50][50], C[50][50], i, x, y, M, N, L;
printf("A: M si N="); scanf("%d %d", &M, &N);  // citire dimensiuni
printf("B: L="); scanf("%d", &L);

for (x=0; x<M; x++)        //citire matrice A (MxN elemente)
  for (y=0; y<N; y++)
  {
    printf("A[%d][%d]=",x,y); scanf("%d", &A[x][y]);
  }

for (x=0; x<N; x++)        //citire matrice B (NxL elemente)
  for (y=0; y<L; y++)
  {
    printf("B[%d][%d]=",x,y); scanf("%d", &B[x][y]);
  }
```

Tablouri de date (continuare)

- **Matrice 2D (continuare)**

```
for (x=0; x<M; x++)      //iniţializare matrice produs
 for (y=0; y<L; y++)
   C[x][y]=0;

for (x=0; x<M; x++)   //calcul produs, parcurgere matrice C
{                     // (MxL elemente) şi afişare
 for (y=0; y<L; y++)
  {
   for (i=0; i<N; i++)
     C[x][y]+=A[x][i]*B[i][y];

   printf("%d ", C[x][y]);
  }
  printf("\n");
}
```


Tablouri de date (continuare)

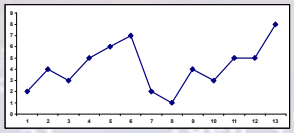
- **Matrice n-dimensionale**

> Am vorbit până acum de matrice uni-dimensionale (vectori) şi matrice bi-dimensionale (tablouri). În realitate, în limbajul C este posibil să definim matrice **n**-dimensionale, cu n>2.

> Ţinând cont că practic nu putem reprezenta grafic date n-dimensionale, cu n>4, **nu toate matricele sunt semnificative** şi îşi au rostul.

- **vectori:**

```
int x[10];
```



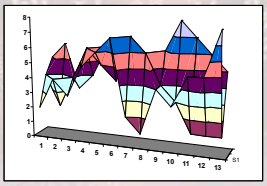
Exemplu: evoluţia temporală a unei mărimi (n=1)

Tablouri de date (continuare)

- **Matrice n-dimensionale (continuare)**

- **matrice 2D:**

```
int x[10][10];
```



Exemplu: o anumită suprafaţă în funcţie de coordonatele spaţiale XoY (n=2)

- **matrice 3D:**

```
int x[10][10][100];
```

Exemplu: evoluţia temporală a imaginilor unui film alb-negru (n=3)

timp

Tablouri de date (continuare)

- **Matrice n-dimensionale (continuare)**

- **matrice 4D:**

```
int x[640][480][3][1500];
```

*eroare: **out of memory***

Exemplu: evoluţia temporală a imaginilor unui film color (n=4)

timp

> În practică este ineficientă reprezentarea statică a matricelor cu dimensiuni mai mari de n=3, datorită necesarului de memorie deloc neglijabil.

→ ex. anterior: **640**x**480**x32x1500 biţi = **1.716** GB (~1 minut de film)

6.2. Lucrul cu şiruri de caractere

Şiruri de caractere

> Limbajul C **nu pune** la dispoziţia utilizatorului un tip individual de date care să permită stocarea şirurilor de caractere (exemplu din alte limbaje: String, AnsiString, etc.)

un şir de caractere = o mulţime ordonată de caractere alfanumerice (string).

Exemplu: "exemplu şir"={'e','x','e','m','p','l','u',' ','ş','i','r'}

Observaţie: constantele şir de caracter sunt marcate cu ghilimele " "

Observaţie: constantele de tip caracter sunt marcate cu apostrof ' '

> Limbajul C permite totuşi lucrul cu şiruri de caractere, dar acestea sunt reprezentate ca **vectori de caractere**.

Şiruri de caracterе (continuare)

> *Modul de alocare al unui şir de caractere:*

```
char x[100];
```

→ variabila **x** reprezintă un şir de caractere ce poate conţine maxim 99 de caractere.

> Principiul alocării memoriei este identic cu cel folosit la vectori, şirul de caractere fiind un vector de tip char.

8 biţi	...	8 biţi	\0
adr. N		*adr. N+98*	*adr. N+99*

> **Atenţie**: un şir de caractere se încheie întotdeauna cu caracterul special '\0' sau NULL (sfârşit de string)

→ dimensiunea vectorului = nr.caractere+1

Şiruri de caracterе (continuare)

> *Modul de citire al şirurilor de caractere*

- o modalitate de citire constă în citirea şirului de caractere ca un vector:

```
char s[5]; int i;
for (i=0;i<4;i++)
  s[i]=getche();
s[i]=NULL;
```

→ **s** este introdus caracter cu caracter prin intermediul funcţiei getche(), la sfârşit se adaugă caracterul NULL.

ineficient !

- modalitatea de citire cea mai uzuală constă în folosirea funcţiei scanf():

```
char s[50];
scanf("%s",s);
```

→ funcţia **scanf** permite citirea şirurilor prin opţiunea %s. Caracterul NULL este adăugat automat la sfârşit.

operatorul & ???

Şiruri de caracterе (continuare)

> *Lucrul cu şirurile de caractere:*

- bineînţeles că toate operaţiile ce pot fi efectuate cu vectori, sunt valabile şi la lucrul cu şirurile de caractere,
- **Atenţie**: şirurile de caractere sunt vectori, astfel:

```
char s[100];
s="proba";
```

→ **eroare**, nu se pot face atribuiri directe şi nici calcule cu şiruri.

- limbajul C pune totuşi la dispoziţia utilizatorului o bibliotecă de funcţii de lucru cu şiruri de caractere, **string.h** :
  - **strcpy** : copiază un şir în altul,
  - **strcmp** : compară două şiruri,
  - **strlen** : determină dimensiune string, etc.

Şiruri de caracterе (continuare)

> *Exemplu: concatenarea a două şiruri.*

```
char s1[100], s2[100], s3[200];
int i, pos;
printf("Introduceti doua siruri:");
scanf("%s %s",s1, s2);
for (i=0; i<strlen(s1); i++)
  s3[i]=s1[i];
pos=i;
for (i=0; i<strlen(s2); i++)
  s3[pos+i]=s2[i];
s3[pos+i]=NULL;
printf("\n%s",s3);
```

citire şiruri s1 şi s2.

se parcurge primul şir şi caracterele sunt adăugate progresiv noului şir s3.

se memorează poziţia ultimului caracter.

se parcurge al doilea şir şi se adaugă caracterele în continuarea lui s3.

sfârşit de string.

6.3. Lucrul cu structuri

Tipuri de date compuse

> La punctele anterioare am vorbit despre tablouri de date: *vectori de date*, *matrice bidimensionale*, *matrice n-dimensionale* şi *şiruri de caractere*.

> Pe lângă reprezentarea datelor sub formă matriceală (date omogene), limbajul de programare **C** mai permite şi lucrul cu **structuri de date**: grupuri de date neomogene (tipuri diferite).

> În acest sens, există tipurile următoare:

- tipul **struct**,
- tipul **union**,

Tipul struct

P *Să se realizeze un program care să permită **stocarea** şi **manipularea** datelor personale a N persoane.*

Exemplu de persoană:

nume – şir de caractere (**char** []),
prenume – şir de caractere (**char** []),
vârstă – număr întreg (**int**),
înălţime – număr real (**float**),
ocupaţie – şir de caractere (**char** []).

x1000 ~ 5000 de variabile indep. !

> Problema este simplă dacă ar fi vorba de o singură persoană, ce facem dacă trebuie să introducem aceste date pentru 1000 de persoane ?

Tipul struct (continuare)

> Soluţia există de mult timp în domeniul bazelor de date, unde astfel de colecţii erau reprezentate pe baza înregistrărilor:

Exemplu bază de date:

Last Name	First Name	Title	Title Of Courtesy	Birth Date	Hire Date	Address
Davolio	Nancy	Sales Representative	Ms.	08-Dec-1968	01-May-1992	507 - 20th Ave. E. Apt. 2A
Fuller	Andrew	Vice President, Sales	Dr.	19-Feb-1952	14-Aug-1992	908 W. Capital Way
Leverling	Janet	Sales Representative	Ms.	30-Aug-1963	01-Apr-1992	722 Moss Bay Blvd.
Peacock	Margaret	Sales Representative	Mrs.	19-Sep-1958	03-May-1993	4110 Old Redmond Rd.
Buchanan	Steven	Sales Manager	Mr.	04-Mar-1955	17-Oct-1993	14 Garrett Hill
Suyama	Michael	Sales Representative	Mr.	02-Jul-1963	17-Oct-1993	Coventry House Miner Rd.
King	Robert	Sales Representative	Mr.	29-May-1960	02-Jan-1994	Edgeham Hollow Winchester Way
Callahan	Laura	Inside Sales Coordinator	Ms.	09-Jan-1958	05-Mar-1994	4726 - 11th Ave. N.E.
Dodsworth	Anne	Sales Representative	Ms.	02-Jul-1969	15-Nov-1994	7 Houndstooth Rd.

se definesc o serie de tipuri de date (variabile)
→ **definite o singură dată**

ceea ce se schimbă sunt valorile pentru fiecare persoană
→ **înregistrări**

Tipul struct (continuare)

> Limbajul C permite manipularea acestor tipuri de date prin folosirea **structurilor**,

structură de date= o colecţie de **variabile**, de regulă, de tipuri diferite, ce se reunesc într-o singură variabilă "container".

Sintaxă:

```
struct <NumeTipNou>
{
 <TipDeDateA> <NumeVariabilă1>;
  ...
 <TipDeDateX> <NumeVariabilăN>;
};
```

- am definit un nou tip de date (structură) numit:<**NumeTipNou**>

- acesta conţine o serie de alte tipuri de date, TipA … TipX.

";" se încheie definiţia.

Tipul struct (continuare)

Exemplu:

```
struct Persoana
{
 char nume[100];
 char prenume[100];
 int varsta;
 float inaltime;
};
```

-am definit tipul de date structura numit: **Persoana**

-acesta va permite stocarea a două şiruri de caractere, a unui int şi a unui număr real.

> Deocamdată am definit doar tipul de date, nu şi o variabilă care să stocheze aceste date, pentru aceasta:

```
struct Persoana OPersoanaAnume;
```

variabila OPersoanaAnume este de tip **Persoana**.

Tipul struct (continuare)

> Declararea variabilelor struct (continuare):

```
struct Persoana OPersoanaAnume;
```

variabila OPersoanaAnume este de tip **Persoana**.

> ***Efect***: în memorie se va aloca spaţiu pentru stocarea a 100+100 de caractere (200x8biţi), a unui int (32 biţi) şi a unui float (32 biţi).

> Aceste locaţii de memorie vor fi adresate prin intermediul variabilei OPersoanaAnume:

OPersoanaAnume
- .nume[100]
- .prenume[100]
- .varsta
- .inaltime

Tipul struct (continuare)

> Citirea şi manipularea variabilelor unei structuri:

```
struct Patrat
{
 int latura;
 float arie;
} P1, P2;
printf("Patrat 1:");
scanf("%d", &P1.latura);
P1.arie=P1.latura*P1.latura;
printf("\nPatrat 2:");
scanf("%d", &P2.latura);
P2.arie=P2.latura*P2.latura;
printf("Suma arii este: %f", P1.arie+P2.arie);
```

- variabilele de tip structură pot fi definite şi **direct** după specificarea tipului.

- elementele sunt adresate prin: **P1.latura** şi **P1.arie**, acestea fiind practic ele însele nişte variabile (fac parte din P1).

- calculele se realizează folosind variabilele, ca şi cum ar fi independente.

Tipul struct (continuare)

Exemplu:

```
struct Patrat
{
 int latura;
 float arie;
};
struct Patrat MultePatrate[100];
int i;
for (i=0;i<100; i++)
{
  scanf("%d", &MultePatrate[i].latura);
  MultePatrate[i].arie=MultePatrate[i].latura*MultePatrate[i].latura;
}
printf("Patratul 10 are aria: %f", MultePatrate[9].arie);
```

-am definit un vector de variabile de tip Patrat (structura), ce va fi stocat în variabila MultePatrate.

-astfel avem 100 de variabile de tip Patrat: MultePatrate[0], MultePatrate[1], … , MultePatrate[99].

Tipul struct (continuare)

Exemplu:

```
struct SPunct
{
 int x,y;
};
int i;
struct Triunghi
{
  struct SPunct puncte[3];
} Triunghi1;
for (i=0; i<3; i++)
{
 printf("punct %d ", i+1);
 scanf("%d %d", &Triunghi1.puncte[i].x, &Triunghi1.puncte[i].y);
}
```

-structura **SPunct** permite stocarea a două valori întregi, şi anume coordonatele x şi y ale unui punct.

-structura **Triunghi** permite stocarea coordonatelor celor trei puncte ce definesc un triunghi prin intermediul vectorului: puncte[3].

din variabila Triunghi1, accesăm elementul puncte[i] (structura) şi mai departe x şi y

Tipul union

> Pe lângă structurile de date, în limbajul C mai există un tip de date similar, şi anume uniunile sau **union.**

> Din punct de vedere al sintaxei şi al modului de declarare al variabilelor, acesta este identic cu tipul **struct**, astfel:

Sintaxă:

```
union <NumeTipNou>
{
 <TipDeDateA> <NumeVariabilă1>;
  ...
 <TipDeDateX> <NumeVariabilăN>;
};
```

- am definit un nou tip de date (uniune) numit:<**NumeTipNou**>

- acesta conţine o serie de alte tipuri de date, TipA … TipX.

";" se încheie definiţia.

Tipul union (continuare)

Exemplu:

```
union VarHibrida
{
 int Ix;
 float Fx;
 double Dx;
 char Cx[8];
} proba;
```

-am definit tipul de date uniune numit: **VarHibrida**

-lista datelor ce pot fi conţinute de acesta,

-variabila proba este de acest tip, şi anume **VarHibrida**.

> ***Efect***: în memorie **se va aloca spaţiu** pentru a putea stoca valoarea variabilei, din lista de variabile a uniunii, ce necesită **spaţiul de memorie cel mai mare**:

int = 32 biţi, float = 32 biţi, double = 64 biţi, char [8] = 64 biţi

Tipul union (continuare)

> Nu se alocă spaţiu pentru toate variabilele, ci doar suficient spaţiu pentru a putea stoca oricare dintre valorile variab.

```
union VarHibrida
{
 int Ix;
 float Fx;
 double Dx;
 char Cx[8];
} proba;
```

coduri ASCII: 8biţi 8biţi ... 8biţi

adr. N

> Astfel, la un moment dat, variabila proba nu va putea avea decât una dintre valorile enumerate: int, float, double sau char.

Exemplu:

```
proba.Ix=10;
proba.Fx=10.5;
scanf("%s",proba.Cx);
```

> Fiind disponibilă doar o locaţie de memorie, valorile sunt suprapuse progresiv pe măsură ce sunt schimbate.

Tipul union (continuare)

Exemplu:

```
union Utest
{
  int x, y, z;
} Var;
Var.y=3; Var.z=20;
printf("%d %d %d", Var.x, Var.y, Var.z);
```

ce se afişează pe ecran?

20 20 20

x, y şi z sunt stocate la aceeaşi locaţie de memorie.

Exemplu:

```
union Utest
{
  int x, z; float y;
} Var;
Var.y=3.3; Var.z=20;
printf("%d %f %d", Var.x, Var.y, Var.z);
```

ce se afişează pe ecran?

20 0.00000 20 **de ce???**

memorie Var

20=00000 10100

int *float* *int*

Sfârşitul Cursului 6