

Universitatea "Politehnica" din București
Facultatea de Electronică, Telecomunicații și
Tehnologia Informației

Programarea Calculatoarelor (limbajul C)

Curs 6 – Tipuri de date compuse

Prof. Bogdan IONESCU

Tablouri de date

> Un tablou este o **structură omogenă** formată dintr-o mulțime finită de elemente de același tip, denumit și tip de bază.

Exemplu:

3.4	3	1.5
2	2.2	1.9
3	2.4	8

- informația este stocată sub formă matriceală
- sursa principală de informație sunt cifrele,
- sursa secundară de informație este dată de relațiile de vecinătate.

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

Curs Programarea Calculatoarelor, Prof. Bogdan IONESCU 3/54

Cuprins

- 6.1. Lucrul cu tablouri
- 6.2. Lucrul cu șiruri de caractere
- 6.3. Lucrul cu structuri

Curs Programarea Calculatoarelor, Prof. Bogdan IONESCU 1/54

Tablouri de date (continuare)

> Informația de vecinătate este esențială și constituie motivul folosirii tablourilor și nu a unei liste de valori.

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

- valori independente în memorie

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

- valori cu vecinătate pe o direcție

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

Curs Programarea Calculatoarelor, Prof. Bogdan IONESCU 4/54

6.1. Lucrul cu tablouri

Curs Programarea Calculatoarelor, Prof. Bogdan IONESCU 2/54

Tablouri de date (continuare)

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

- valori cu vecinătate pe două direcții

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 informațiilor suplimentare.

imagine cu două obiecte

Curs Programarea Calculatoarelor, Prof. Bogdan IONESCU 5/54

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)**
- **șiruri de caractere**

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

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 locații de memorie consecutive de tip **int**, ce vor corespunde valorilor vectorului x:

32 biți	32 biți	32 biți	32 biți
adr. N	adr. N+1	adr. N+2	adr. N+3

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

Tablouri de date (continuare)

• Vectori

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

Exemplu: $\begin{bmatrix} 1 & 3 & 9 & 7 \\ 0 & 1 & 2 & 3 \end{bmatrix}$ sau $\begin{bmatrix} 0 & 1 \\ 1 & 3 \\ 2 & 9 \\ 3 & 7 \end{bmatrix}$

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

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

Atenție: **indicele** primului element este întotdeauna 0.

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 locație de memorie a vectorului x (456).

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

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

Tablouri de date (continuare)

• Vectori (continuare)

> Modul de definire a unui vector în limbajul C:

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

tipul datelor conținute 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)

> Exemplu citire vector:

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

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

Execuție:

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

> **Observație:** 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, și citește dimensiunea dorită (<200).

citește elementele vectorului.

maxim este primul element.

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

Tablouri de date (continuare)

• Vectori (continuare)

P

Enunț: să se concateneze doi vectori de numere întregi de dimensiuni N și respectiv M. Vectorul obținut va fi stocat separat (N,M<100).

Structură program:

Variabile de intrare/lucru:

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

Variabile de ieșire:

```
int v[200];
```

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

Tablouri de date (continuare)

• Vectori (continuare)

P

Enunț: să se calculeze suma a doi vectori de numere reale. Vectorii au același 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 ieșire:

```
float suma;
```

Structură program:

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

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 și profităm de for și stocăm valori în v
{
    printf("v1[%d]=", i); scanf("%d", &v1[i]);
    v[i]=v1[i];
}
for (i=0; i<M; i++) // citim v2 și profităm de for și 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)

• 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]; // inițializăm suma cu primele elemente
for (i=1; i<dim; i++)
    suma+=v1[i]+v2[i];
printf("suma este: %.2f", suma);
```

Tablouri de date (continuare)

• Matrice 2D

> O matrice 2D reprezintă o colecție de date de același tip ce este structurată pe linii și coloane (sub formă de tablou)

Exemplu:

	0	1	2	
0	2	9	1	linie ↓ coloană
1	1	7	2	
2	4	5	3	

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

Observație: 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.colioane>];
```

tipul datelor
conținute de matrice
ex.: int, char, float

numele variabilei
de tip matrice
(identificator)

numărul **maxim**
de linii și coloane
ale matricei

Exemple:

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

- variabila **x** este o matrice ce conține 100 de valori *intregi*, pe 10 linii și 10 coloane
- variabila **y** este o matrice ce conține 50 de valori *reale*, pe 5 linii și 10 coloane.

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

Observație: practic o matrice este o colecție de vectori de linie, în cazul de față avem doi vectori de adrese N și N+4.

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

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

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 locații de memorie consecutive de tip **int**, ce vor corespunde liniilor și coloanelor matricei **m**:

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

> Observație: adresele sunt de regulă consecutive.

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 și coloana de indice 3

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, și anume a primului element al acesteia (N+4),

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]);
}
```

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

Execuție:

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

> Observație: ca și în cazul vectorilor, nu este obligatoriu să folosesc toate locațiile 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], NI, Nc;
int i, j, prod=1;
printf("NI si Nc:");
scanf("%d %d", &NI, &Nc);
for (i=0; i<Nc; i++)
for (j=0; j<NI; 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, și citește nr. linii (<5) și nr. coloane (<10).

citește elementele matricei.

calculează produsul elementelor matricei v.

Este o greșeală importantă în program!

Tablouri de date (continuare)

• Matrice 2D (continuare)

P

Enunț: să se realizeze produsul a două matrice de numere întregi de dimensiuni diferite, MxN și NxL.

Variabile de intrare/lucru:

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

Variabile de ieșire:

```
int C[50][50];
```

Problemă:

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

Tablouri de date (continuare)

• Matrice 2D (continuare)

P

Enunț: să parcurgă elementele de sub diagonală principală (inclusiv) ale unei matrice pătratică de numere reale (<20x20).

```
2 9 1
1 7 2
4 5 3
```

Variabile de intrare/lucru:

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

Variabile de ieșire:

valori matrice,

Structură program:

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

Tablouri de date (continuare)

• Matrice 2D (continuare)

> Produsul a două matrice:

$$\begin{matrix} N & & & & L \\ A & \times & B & = & C \\ M & \begin{bmatrix} 2 & 9 & 1 \\ 1 & 7 & 2 \\ 4 & 5 & 3 \end{bmatrix} & \begin{bmatrix} 4 & 5 \\ 2 & 1 \\ 1 & 8 \end{bmatrix} & = & \begin{bmatrix} -0,0 & -0,1 \\ -1,0 & -1,1 \\ -2,0 & -2,1 \end{bmatrix} \end{matrix}$$



$$C[1][0] = A[1][0] \cdot B[0][0] + A[1][1] \cdot B[1][0] + A[1][2] \cdot B[2][0]$$

>Formula magică este:

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

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");
```

```
}
```

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

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

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

Curs Programarea Calculatoarelor, Prof. Bogdan IONESCU

30/54

Tablouri de date (continuare)

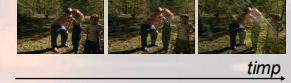
• Matrice n-dimensionale (continuare)

• matrice 4D:

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

eroare: **out of memory**

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



time

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

→ ex. anterior: **640x480x32x1500** biți =
1.716 GB (~1 minut de film)

Curs Programarea Calculatoarelor, Prof. Bogdan IONESCU

33/54

Tablouri de date (continuare)

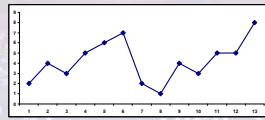
• Matrice n-dimensionale

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

> Ținând cont că practic nu putem reprezenta grafic date n-dimensionale, cu $n > 4$, **nu toate matricele sunt semnificative** și își au rostul.

• vectori:

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



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

Curs Programarea Calculatoarelor, Prof. Bogdan IONESCU

31/54

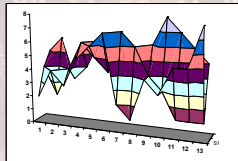
6.2. Lucrul cu șiruri de caractere

Tablouri de date (continuare)

• Matrice n-dimensionale (continuare)

• matrice 2D:

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



Exemplu: o anumită suprafață în funcție de coordonatele spațiale XoY (n=2)

• matrice 3D:

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

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



time

Curs Programarea Calculatoarelor, Prof. Bogdan IONESCU

32/54

Șiruri de caractere

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

un șir de caractere = o mulțime ordonată de caractere alfanumerice (string).

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

Observație: constantele șir de caracter sunt marcate cu ghilimele "

Observație: constantele de tip caracter sunt marcate cu apostrof '

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

Curs Programarea Calculatoarelor, Prof. Bogdan IONESCU

35/54

Șiruri de caractere (continuare)

> Modul de alocare al unui șir de caractere:

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

→ variabila **x** reprezintă un șir de caractere ce poate conține maxim 99 de caractere.

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



> **Atenție:** un șir de caractere se încheie întotdeauna cu caracterul special '\0' sau NULL (sfârșit de string)

→ dimensiunea vectorului = nr.caractere+1

Șiruri de caractere (continuare)

> Exemplu: concatenarea a două șiruri.

```
char s1[100], s2[100], s3[200];  
int i, pos;  
printf("Introduceți 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 șiruri s1 și s2.

se parcurge primul șir și caracterele sunt adăugate progresiv noului șir s3.

se memorează poziția ultimului caracter.

se parcurge al doilea șir și se adaugă caracterele în continuarea lui s3.

sfârșit de string.

Șiruri de caractere (continuare)

> Modul de citire al șirurilor de caractere

- o modalitate de citire constă în citirea șirului 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 funcției `getche()`, la sfârșit se adaugă caracterul NULL.

ineficient !

operatorul & ???

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

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

→ funcția **scanf** permite citirea șirurilor prin opțiunea %s. Caracterul NULL este adăugat automat la sfârșit.

6.3. Lucrul cu structuri

Șiruri de caractere (continuare)

> Lucrul cu șirurile de caractere:

- bineînțeles că toate operațiile ce pot fi efectuate cu vectori, sunt valabile și la lucrul cu șirurile de caractere,

- Atenție:** șirurile de caractere sunt vectori, astfel:

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

→ **eroare**, nu se pot face atribuiri directe și nici calcule cu șiruri.

- limbajul C pune totuși la dispoziția utilizatorului o bibliotecă de funcții de lucru cu șiruri de caractere, **string.h** :

- **strcpy** : copiază un șir în altul,
- **strcmp** : compară două șiruri,
- **strlen** : determină dimensiune string, etc.

Tipuri de date compuse

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

> Pe lângă reprezentarea datelor sub formă matriceală (date omogene), limbajul de programare **C** mai permite și 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** și **manipularea** datelor personale a **N persoane**.

Exemplu de persoană:



nume – șir de caractere (**char []**),
prenume – șir de caractere (**char []**),
vârstă – număr întreg (**int**),
înălțime – număr real (**float**),
ocupație – șir 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)

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 șiruri de caractere, a unui int și a unui număr real.

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

```
struct Persoana OPersoanaAname;
```

variabila
 OPersoanaAname
 este de tip **Persoana**.

Tipul struct (continuare)

> Soluția există de mult timp în domeniul bazelor de date, unde astfel de colecții 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
Leventing	Janet	Sales Representative	Ms.	30-Aug-1963	01-Apr-1992	722 Moss Bay Blvd.
Peacock	Margaret	Sales Representative	Ms.	15-Sep-1956	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-1983	17-Oct-1993	Coventry House Miner Rd.
King	Robert	Sales Representative	Mr.	29-May-1960	02-Jan-1994	Edgeham Hollow Winchester Way
Callahan	Laura	Wide Sales Coordinator	Ms.	09-Jan-1958	05-Mar-1994	4726 - 11th Ave. NE
Dodsworth	Anne	Sales Representative	Ms.	02-Jul-1969	15-Nov-1994	7 Houndgheath 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)

> Declararea variabilelor struct (continuare):

```
struct Persoana OPersoanaAname;
```

variabila
 OPersoanaAname
 este de tip **Persoana**.

> **Efect:** în memorie se va aloca spațiu pentru stocarea a 100+100 de caractere (200x8biți), a unui int (32 biți) și a unui float (32 biți).

> Aceste locații de memorie vor fi adresate prin intermediul variabilei OPersoanaAname:

```
OPersoanaAname
├── .nume[100]
├── .prenume[100]
├── .varsta
└── .inaltime
```

Tipul struct (continuare)

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

structură de date= o colecție 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 conține o serie de alte tipuri de date, TipA ... TipX.

„,” se încheie definiția.

Tipul struct (continuare)

> Citirea și 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 și **direct** după specificarea tipului.

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

- calculele se realizează folosind variabilele, ca și 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 union (continuare)

Exemplu:

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

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

-lista datelor ce pot fi conținute de acesta,

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

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

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

Tipul struct (continuare)

Exemplu:

```
struct SPunct
{
    int x,y;
};

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, și anume coordonatele x și 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) și mai departe x și y

Tipul union (continuare)

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

coduri ASCII: 8biți 8biți ... 8biți
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.lx=10;
proba.Fx=10.5;
scanf("%s", proba.Cx);
```

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

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

Tipul union

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

> Din punct de vedere al sintaxei și 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 conține o serie de alte tipuri de date, TipA ... TipX.

„,” se încheie definiția.

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 afișează pe ecran?

20 20 20

x, y și z sunt stocate la aceeași locație 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 afișează pe ecran?

20 0.00000 20

de ce???

memorie Var

20=00000 10100

int float int

Sfârșitul Cursului 6