

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

1 1 0 0 0 1 0 1
 1 1 0 1 0 1 0 1


Structuri de Date și Algoritmi (limbajul C)

Curs 2 – Structuri simple de date

Prof. Bogdan IONESCU

2015-2016

Cuprins

- 2.1. Lucrul cu structuri
- 2.2. Lucrul cu uniuni

Curs Structuri de Date și Algoritmi, Prof. Bogdan IONESCU, 2015-2016

1/33

2.1. Lucrul cu structuri


Curs Structuri de Date și Algoritmi, Prof. Bogdan IONESCU, 2015-2016

2/33

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 ?

Curs Structuri de Date și Algoritmi, Prof. Bogdan IONESCU, 2015-2016

3/33

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
Leverling	Janet	Sales Representative	Ms.	30-Aug-1963	01-Apr-1992	722 Moss Bay Blvd.
Peterson	Margaret	Sales Representative	Mrs.	19-Sep-1958	03-May-1993	4110 Olkt. Redmond Bld.
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**

Curs Structuri de Date și Algoritmi, Prof. Bogdan IONESCU, 2015-2016

4/33

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.

Curs Structuri de Date și Algoritmi, Prof. Bogdan IONESCU, 2015-2016

5/33

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 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 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 OPersoanaAnume:

```
OPersoanaAnume
├── .nume[100]
├── .prenume[100]
└── .varsta
```

Tipul struct (continuare)

> Citirea și manipularea variabilelor unei structuri:

```
struct Patrat
{
    int latura;
    float arie;
} P1, P2;

scanf("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 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, 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 struct (continuare)

P **Enunț:** să se realizeze un program ce permite stocarea numelui, prenumelui, vârstei și genului pentru un grup de persoane. Citirea datelor și afișarea acestora se vor realiza cu funcții separate.

Variabile de intrare/lucru:

Variabile de ieșire:

```
struct DatePersoana {...};
struct DatePersoana Grupa411A[40];
int n;
```

Funcții:

```
void CitireDate(struct DatePersoana Grupa[40], int n);
void AfișareDate(struct DatePersoana Grupa[40], int n);
```

Tipul struct (continuare)

```
struct DatePersoana
{
    char nume[256], prenume[256];
    int varsta;
    enum tipgen {masculin,feminin} gen;
} Grupa411A[40];

void CitireDate(struct DatePersoana Grupa[40], int n)
{
    for (int i=0;i<n;i++)
    { printf("\n#Citire date persoana %d\n",i);
      printf("Nume:");scanf("%s",Grupa[i].nume);
      printf("Prenume:");scanf("%s",Grupa[i].prenume);
      printf("Varsta:");scanf("%d",&Grupa[i].varsta);
      printf("Gen (0-masculin, 1-feminin):");scanf("%d",&Grupa[i].gen);}
}
```

Tipul struct (continuare)

```
void AfisareDate(struct DatePersoana Grupa[40], int n)
{
    printf("\n#Date persoane\n");
    for (int i=0;i<n;i++)
    { printf("%s %s, %d ani", Grupa[i].nume,Grupa[i].prenume,
      Grupa[i].varsta);
      if (!Grupa[i].gen) printf(", gen masculin\n");
      else printf(", gen feminin\n"); }
}

int main ()
{
    int n;
    printf("numar persoane:"); scanf("%d",&n);
    CitireDate(Grupa411A, n);
    AfisareDate(Grupa411A, n);
}
```

Tipul struct (continuare)

P *Enunț:* să se realizeze un program ce permite stocarea coordonatelor punctelor unui poligon. Tipul poligonului (numărul de puncte) este specificat de utilizator.

Variabile de intrare/lucru:

```
struct Punct (int x,y);
struct Poligon {int numar_puncte,
                struct Punct PunctePoligon[100]};
struct Poligon UnPoligon;
```

Variabile de ieșire:

-



Tipul struct (continuare)

```
struct Punct
{
    float x,y; //definire tip Punct ce stocheaza coordonate x si y
};

struct Poligon //definire tip Poligon ce contine
{ //nr puncte
    int nr; //stocate intr-un vector de tip Punct
    struct Punct PunctePoligon[100]; //fiecare valoare avand un x și y
};

struct Poligon UnPoligon; //definire variabila de tip Poligon
```

Tipul struct (continuare)

```
int main ()
{
    printf("numar de puncte:");scanf("%d",&UnPoligon.nr);
    printf("Introduceti coordonate poligon:\n");
    for (int i=0;i<UnPoligon.nr;i++)
    {
        printf("punct%d-x:",i); scanf("%f",&UnPoligon.PunctePoligon[i].x);
        printf("punct%d-y:",i); scanf("%f",&UnPoligon.PunctePoligon[i].y);
    }

    printf("\nCoordonate poligon (%d puncte)\n",UnPoligon.nr);
    for (int i=0;i<UnPoligon.nr;i++)
        printf("%.2f,%.2f\n",UnPoligon.PunctePoligon[i].x
            ,UnPoligon.PunctePoligon[i].y);
}
```

Tipul struct (continuare)

> Dacă se dorește modificarea valorilor unei structuri prin intermediul funcțiilor, este necesară transmiterea acestora prin adresă (vezi vectori).

> să reluăm problemele anterioare ...

Problemă stocare date persoane:

```
void CitireDate(struct DatePersoana Grupa[40], int n);
```

→ funcția primește la intrare o variabilă structură pe care o modifică. *De ce modificările sunt vizibile și în programul care o apelează?* (sunt permanente)

De fapt structura este trimisă prin *adresă* fiind un vector de structuri, unde **Grupa** este adresa primului element.

Tipul struct (continuare)

> Pointeri la structuri:

```
struct Punct {int x,y;} UnPunct;
```

Punct este un tip structură ce poate stoca două valori **int**, iar **UnPunct** este o variabilă de acest tip;

UnPunct

32biți 32biți
adr.3234

```
struct Punct *PUnPunct;
```

PUnPunct este un pointer la o variabilă de tip structură Punct (conform definiție pointeri: <tip date> *<nume pointer>)

UnPunct **PUnPunct**

32biți 32biți ?!#
adr.3234 adr.2346

Curs Structuri de Date și Algoritmi, Prof. Bogdan IONESCU, 2015-2016 18/33

Tipul struct (continuare)

> Pointeri la structuri (continuare):

> **Atenție:** orice variabilă pointer trebuie inițializată cu o adresă validă a unei date de același tip indicat:

```
PUnPunct=&UnPunct;
```

PUnPunct este inițializat cu adresa lui **UnPunct**;

UnPunct **PUnPunct**

32biți 32biți 3234
adr.3234 adr.2346

> O structură conține mai multe date, cum indicăm prin intermediul pointerului conținutul locației (variabilei) dorite?

Curs Structuri de Date și Algoritmi, Prof. Bogdan IONESCU, 2015-2016 19/33

Tipul struct (continuare)

> Pointeri la structuri (continuare):

```
struct Punct {int x,y;} UnPunct;
struct Punct *PUnPunct;
PUnPunct=&UnPunct;
```

O structură conține mai multe date, cum indicăm prin intermediul pointerului conținutul locației (variabilei) dorite?

> prin variabila UnPunct:

UnPunct → .x
 → .y

> prin variabila pointer PUnPunct:

PUnPunct → ->x
 → ->y

Curs Structuri de Date și Algoritmi, Prof. Bogdan IONESCU, 2015-2016 20/33

Tipul struct (continuare)

> Pointeri la structuri (continuare):

```
struct Punct {int x,y;} UnPunct;
struct Punct *PUnPunct;
PUnPunct=&UnPunct;
```

~ UnPunct.x = 10

~ UnPunct.y = 15

```
PUnPunct->x=10;
PUnPunct->y=15;
```

```
printf("x=%d\n",UnPunct.x);
printf("y=%d",UnPunct.y);
```

> x=10
y=15

> Toate operațiile cu pointeri sunt valabile și în acest caz inclusiv alocarea dinamică a memoriei.

Curs Structuri de Date și Algoritmi, Prof. Bogdan IONESCU, 2015-2016 21/33

Tipul struct (continuare)

P **Enunț:** să se realizeze un program ce permite stocarea coordonatelor punctelor unui poligon. Tipul poligonului (numărul de puncte) este specificat de utilizator. Citirea datelor se va realiza într-o funcție.

Variabile de intrare/lucru: *Variabile de ieșire:*

```
struct Punct {int x,y};
struct Poligon {int numar_puncte,
                struct Punct PunctePoligon[100]};
struct Poligon UnPoligon;
struct Poligon *PUnPolygon;
```

Funcții:

```
void CitirePoligon(struct Poligon *UnPoligon);
```

Curs Structuri de Date și Algoritmi, Prof. Bogdan IONESCU, 2015-2016 22/33

Tipul struct (continuare)

```
struct Punct
{
float x,y; //definire tip Punct ce stocheaza coordonate x și y
};

struct Poligon //definire tip Poligon ce contine
{
//nr puncte
int nr; //stocate intr-un vector de tip Punct
struct Punct PunctePoligon[100]; //fiecare valoare avand un x și y
};

struct Poligon UnPoligon; //definire variabila de tip Poligon
struct Poligon *PUnPolygon=&UnPoligon; //initializare pointer
```

Curs Structuri de Date și Algoritmi, Prof. Bogdan IONESCU, 2015-2016 23/33

Tipul struct (continuare)

```
void CitirePoligon(struct Poligon *UnPoligon)
{
do
{
// citire numar puncte
printf("numar de puncte:");
scanf("%d",&UnPoligon->nr);
} while (UnPoligon->nr<3);

printf("Introduceti coordonate poligon:\n");
for (int i=0;i<UnPoligon->nr;i++)
{
printf("punct%d-x: ",i);scanf("%f",&UnPoligon->PunctePoligon[i].x);
printf("punct%d-y: ",i);scanf("%f",&UnPoligon->PunctePoligon[i].y);
}
}
```

Tipul struct (continuare)

```
int main ()
{
CitirePoligon(PUnPoligon);

printf("\nCoordonate poligon (%d puncte)\n",UnPoligon.nr);

for (int i=0;i<UnPoligon.nr;i++)
printf("%.2f,%.2f\n",UnPoligon.PunctePoligon[i].x
,UnPoligon.PunctePoligon[i].y);
}
```

2.2. Lucrul cu uniuni

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 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 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 (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

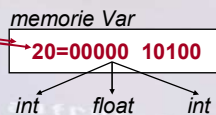
x, y și z sunt stocate la aceeași locație de memorie.

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???



Tipul union (continuare)



Enunț: folosind uniuni să se realizeze un program ce permite codarea unui număr întreg prin inversarea celor patru bytes ai acestuia:



```
383813196 → 00010110 11100000 10000110 01001100
           → 01001100 10000110 11100000 00010110
           → 1283907606 (cod)
```

Variabile de intrare/lucru:

```
union codare {unsigned int i; char bytes[4]};
char tmp; //săptămânar
```

Variabile de ieșire:

-

Tipul union (continuare)

```
char tmp;
union codare
{ unsigned int i;
  char bytes[4]; } cod;

printf("Introduceți numărul de codat:");
scanf("%u", &cod.i);

printf("#codare#\n");
tmp = cod.bytes[0]; cod.bytes[0] = cod.bytes[3]; cod.bytes[3] = tmp;
tmp = cod.bytes[1]; cod.bytes[1] = cod.bytes[2]; cod.bytes[2] = tmp;
printf("codul este: %u\n", cod.i);

printf("#decodare#\n");
tmp = cod.bytes[0]; cod.bytes[0] = cod.bytes[3]; cod.bytes[3] = tmp;
tmp = cod.bytes[1]; cod.bytes[1] = cod.bytes[2]; cod.bytes[2] = tmp;
printf("numărul decodat este: %u", cod.i);
```

Sfârșitul Cursului 2