**Stucturi. Tipul struct**

De multe ori datele folosite într-un program caracterizează obiecte/fenomene din realitate, iar acestea au mai multe caracteristici decât un simplu număr sau un șir de caractere. De exemplu, o persoană poate fi caracterizată prin câteva informații:

* nume – șir de caractere
* prenume – șir de caractere
* sex – (M/F) - caracter
* vârsta – număr natural
* salariu – număr real
* etc.

Pentru a gestiona informațiile despre o persoană avem nevoie de (cel puțin) 5 variabile, câte una pentru fiecare informație caracteristică a persoanelor: nume, prenume, etc., iar dacă am prelucra informații despre n persoane am avea nevoie de 5 vectori cu n elemente, corespunzătoare acelor persoane. Elementele vectorilor trebuie sincronizate, ceea ce devine foarte complicat și obositor. Ar fi mult mai simplu dacă am putea grupa toate aceste informații specifice unei persoane într-o singură variabilă.

Acest lucru este posibil în C/C++, folosind tipul de date struct, prin care vom defini o **structură**. O dată de tip structură înglobează mai multe date componente, numite **câmpuri**, care pot fi de tipuri diferite. Câmpurile pot fi de orice tip cu excepţia tipului fişier.

**Obs.** Tipul struct este un tip de date **neomogen/eterogen** (spre deosebire de tipurile studiate până acum – tablouri, şiruri de caractere),deoarece poate îngloba în definţia lui date de tipuri diferite.

* **Declararea unei structuri**

Sintaxa generală a declarării unui tip de date struct este următoarea:

struct [NUME\_STRUCTURĂ]

{ [TIP1 NUME\_CÂMP[, NUME\_CÂMP[, ...]];]

[TIP2 NUME\_CÂMP[, NUME\_CÂMP[, ...]];]

...

} [LISTA DE VARIABILE];

În acest mod s-a declarat un **tip de date** cu numele NUME\_STRUCTURA și opţional niște variabile de acest tip. În continuare NUME\_STRUCTURA poate fi folosit ca un tip de date oarecare, de exemplu pentru a declara variabile sau în lista parametrilor formali ai unei funcții.

Pentru exemplul dat mai sus, declararea va fi:

struct persoana

{ char nume[21], prenume[21], sex;

int varsta;

double salariu;

};

S-a declarat o structură cu numele persoana, dar nu s-a declarat nicio variabilă de acest tip. Pentru a declara variabile de tip persoana putem să le adăugăm la declararea structurii:

struct persoana

{ char nume[21], prenume[21], sex;

int varsta;

double salariu;

} A, B;

(am declarat odată cu tipul de date persoana și două variable A și B de acest tip), sau putem să le declarăm ulterior declarării tipului de date, astfel:

Această declarare este corectă în C++, dar nu este corectă în C:

persoana A, B;

sau

Această declarare este corectă atât în C++ cât și în C:

struct persoana A, B;

În declarările de mai sus avem următoarele:

* persoana – este numele structurii;
* A și B – sunt variabile de tip persoana sau, mai explicit, struct persoana;
* nume, prenume, sex, varsta, salariu – sunt numele câmpurilor structurii persoana. Ele pot fi utilizate într-un program doar dacă există variabile de tip persoana!

**Obs.** Numele structurii este opţional, deci putem declara tipuri anonime (fără nume), dar în acest caz nu mai putem declara variabile ulterior, după definirea structurii.

**Exemplu**:

struct

{ char nume[21], prenume[21], sex;

int varsta;

double salariu;

} A, B; // toate variabilele trebuie declarate la definirea tipului struct

* **Referirea unui câmp al unei structuri**

Se face cu ajutorul operatorului punct . , numit și operator de acces la membrii unei structuri. Forma generală a acestei operații este:

variabila\_de\_tip\_structura.câmp

**Exemplu**:

Considerăm următoarea secvență:

struct persoana

{ char nume[21], prenume[21], sex;

int varsta;

double salariu;

};

persoana A, B;

Următoarele referiri sunt corecte:

A.nume

A.prenume

A.sex

A.salariu

A.varsta

Următoarele referiri sunt greșite. **DE CE?**

persoana.nume

persoana.A

varsta.persoana

* **Citirea și afișarea datelor de tip struct**

O variabilă de tip structură nu poate fi nici citită, nici afișată.

Citirea, afişarea şi prelucrarea datelor de tip struct se poate face numai câmp cu câmp!

CORECT!

cin >> A.nume >> A.prenume >> A.sex >> A.varsta >> A.salariu;

cout << A.nume <<" "<< A.prenume <<" "<< A.sex <<" "<< A.varsta <<" "<< A.salariu;

GREȘIT!

cin >> A;

cout << B;

* **Operaţii cu structuri**

Singura operație permisă la nivel de structură este atribuirea între două date de acelaşi tip structură.

**Exemplu**:

A = B

// fiecare câmp al variabilei B se copiază în câmpul corespunzător al variabilei A

**Obs.** Cu câmpurile unei structuri se pot face toate operaţiile permise de tipul câmpului respectiv.

Următoarele operații sunt corecte:

cin >> A.varsta;

strcpy(A.nume, "Popescu");

strcpy(B.nume, A.nume);

A.sex = 'M';

A.salariu = B.salariu + 125.80;

B.varsta ++;

* **Structuri imbricate**

O structură poate avea câmpuri de alt tip structură.

**Exemplu:**

struct data

{ int zi, luna, an; };

struct persoana

{ char nume[21], prenume[21], sex;

int varsta;

**data data\_angajarii;**

double salariu;

} A;

Am declarat doua structuri: data – pentru a memora o dată calendaristică, persoana – pentru a memora informații despre o persoană și o variabilă A de tip persoana. Variabila A are un câmp de tip data, numit data\_angajarii. Prin intermediul acestuia avem acces la anul, luna și ziua angajării persoanei respective.

**sau**

struct persoana

{ char nume[21], prenume[21], sex;

int varsta;

**struct // nu mai este necesar numele structurii!**

**{ int zi, luna, an; } data\_angajarii;**

double salariu;

} A;

Următoarele referiri sunt corecte.

A.data\_angajarii // de tip structură

A.data\_angajarii.zi // ziua angajării - de tip int

A.data\_angajarii.luna // luna angajării - de tip int

A.data\_angajarii.an // anul angajării - de tip int

Următoarele referiri sunt greșite. **DE CE?**

A.data

A.data.zi

A.an.data\_angajarii

A.salariu.an

* **Tablouri de structuri. Structuri cu câmpuri de tip tablou**

În C/C++ se pot declara tablouri cu elemente de tip structură și, de asemenea, o structură poate avea elemente de tip tablou (am văzut deja acest lucru în cazul tipului persoana, cu câmpurile nume și prenume de tip șir de caractere).

**Exemplu:**

struct elev

{ char nume[21], prenume[21];

int n1, n2; // notele pe cele două semestre

float med; // media anuală

} **elev v[30];**

int n;

int main()

{ **//citirea vectorului de structuri (elevi) cu calculul mediei fiecăruia**

cin>>n;

for(int i = 1 ; i <= n ; i ++)

{

cin >> v[i].nume >> v[i].prenume >> v[i].n1 >> v[i].n2;

v[i].med = (float) (v[i].n1 + v[i].n2) / 2;

}

**// afişarea vectorului de structuri (elevi)**

for(int i = 1 ; i <= n ; i ++)

cout<<v[i].nume<<" "<<v[i].prenume<<" "<<v[i].n1<<" "<<v[i].n2<<" "<<v[i].med<<endl;

return 0;

}

* **Funcții care lucrează cu structuri**

**De reținut:**

* o funcție C++ poate avea parametri de tip structură
* dacă dorim ca funcția să întoarcă un parametru modificat (cu câmpurile modificate), acesta trebuie transmis prin referință
* o funcție poate returna o valoare de tip structură

**Exemplu:**

Structura punct memorează coordonatele întregi ale unui punct din plan:

struct punct

{ int x,y; } ;

Funcția double Dist(punct P) determină distanța de la origine la punctul P. Parametru este transmis prin valoare:

double Dist(punct P)

{ return sqrt(P.x \* P.x + P.y \* P.y); }

Funcția void Citire(punct &P) citește coordonatele punctului P și întoarce coordonatele:

void Citire(punct &P)

{ cin >> P.x >> P.y; }

Funcția void Afisare(punct P) afişează coordonatele punctului P:

void Afisare(punct P)

{ cout << P.x << " " << P.y << endl; }

Funcția punct Simetric(punct P) returnează o structură cu coordonatele simetricului lui P faţă de origine.

punct Simetric(punct P)

{ punct tmp;

tmp.x = - P.x; tmp.y = - P.y;

return tmp;

}

**Link-uri utile:**

<https://www.pbinfo.ro/articole/7653/tipul-struct>

<https://www.pbinfo.ro/articole/7654/operatii-cu-structuri>

<https://www.pbinfo.ro/articole/7655/structuri-imbricate>

<https://www.pbinfo.ro/articole/7656/structuri-tablouri-si-functii>

**Tema (Pbinfo): 922, 923, 2004, 1013, 3164, 2802**