

Limbaje de Programare

Curs 9 – Tipuri definite de utilizator

Dr. Casandra Holotescu

Universitatea Politehnica Timișoara

Ce discutăm azi...

- ① Tipuri definite de utilizator
- ② Tipul enumerare
- ③ Tipul structura
- ④ Tipul uniune

Tipuri

Un **tip** definește:

- o **mulțime** de valori
- operațiile** posibile cu acestea

În C putem:

redenumi tipuri existente, pentru mai multă claritate
defini **tipuri noi**: **enumerare, structură, uniune.**

Declararea de tipuri

Se face folosind cuvântul cheie **typedef**:

typedef nume-tip-existent nume-tip-nou;

Se folosește deseori pentru creșterea clarității și a lizibilității programului.

```
typedef double real;  
// real devine un alt nume pt. tipul double  
..  
real x = 3.14; // folosim tipul declarat
```

Declararea de tipuri – exemple

```
typedef int intreg;
typedef int *pintreg;
typedef int **ppintreg;
```

...

```
intreg a = 5;
pintreg pa = &a;
ppintreg ppa = &pa;
printf("%d\n", **ppa);
```

Tipul enumerare

Tipul enumerare dă nume unui **șir de valori** numerice cu o anumită semnificație ⇒ crește lizibilitatea programului

```
enum nume-enum { nume-val1, nume-val2, .. };
```

```
enum luniReci {ian=1, feb, mar, nov=11, dec};
```

definește un tip cu numele *enum luniReci* (cuvântul cheie **enum** face parte din numele tipului!)

enum luniReci definește următoarele asocieri nume-valori:

ian = 1, feb = 2, mar = 3, nov = 11, dec = 12

Tipul enumerare

Tipul enumerare este un **tip întreg!**

⇒ valorile de tip enumerare se folosesc ca întregi

Implicit, sirul valorilor e crescător, începând de la 0.

Putem specifica și explicit valori (vezi *nov=11*).

O valoare se poate repeta.

Un nume de constantă nu se poate folosi în mai multe enumerări.

Exemplu – tipul enumerare

```
enum zile {D, L, Ma, Mc, J, V, S};  
// declara tipul enum zile  
enum zile zi;  
// declara variabila zi de tip zile  
  
int nr_ore_lucru[7];  
// numar de ore pe zi  
  
for (zi = L; zi <= V; zi++)  
    nr_ore_lucru[zi] = 8;
```

Tipul structură

Un **tip structură** grupează împreună mai multe **elemente de tipuri diferite**, legate între ele de o semnificație comună.

```
struct nume-struct {  
    nume-tip1 nume-camp1;  
    nume-tip2 nume-camp2;  
    ...  
    nume-tipn nume-campn;  
};
```

definește un tip cu numele *struct nume-struct*

Elementele unei structuri se numesc **câmpuri**.

Pot fi de orice tip, dar **nu** de **același** tip structură
(nu tip recursiv, ar fi nevoie de memorie infinită!).

Numele câmpurilor sunt vizibile **doar în interiorul** structurii!

Tipul structură

```
struct student {  
    // numele complet al tipului (incl. "struct")  
  
    char nume[32], prenume[32];  
    // doua tablouri de caractere  
    char *domiciliu;  
    // ADRESA! memoria pt. sir se aloca altundeva  
    float medie_an[4];  
    float nota_dipl;  
} stud1, stud2;  
// doua variabile declarate direct aici  
  
struct student s3;  
// declaratie separata de variabila
```

Folosirea structurilor

Variabilele/valorile de tip structură pot fi:

- atribuite
- date ca parametru la funcții
- returnate ca rezultat

Accesul unui câmp al unei structuri se face prin intermediul **operatorului de selectie .** (punct)

nume-var.nume-camp

```
struct student s3;  
s3.nota_dipl = 9.25;
```

Structurile **nu pot** fi comparate cu operatori logici!

Se compară individual câmpurile lor.

Folosirea structurilor – exemple

```
struct point {
    float x, y;
} pct1 = { 2.5, 1.5 };
// initializare intre accolade, ca la tablouri

struct student s, p; // declara variab s,p
strcpy(s.nume, "Stefanovici");
// NU! s.nume = ... (s.nume e un tablou)

s.domiciliu = "str. Linistei nr. 2";
// sau alocare dinamica (malloc) si strcpy

s.medie_an[2] = 9.35; //ca orice variabila
p = s; // atribuire intre structuri
```

Structuri și **typedef**

Tipurile structura pot fi redenumite folosind **typedef**, pentru a fi utilizate mai usor în program.

```
typedef struct student {  
    /* ceva campuri */  
} student_t;
```

Sau, având separat definirea tipului și redenumirea lui:

```
struct student {  
    /* ceva campuri */  
}; // definește tipul  
  
typedef struct student student_t;  
// declara un nou nume pt. struct student
```

Pointeri la structuri

Pentru eficiență, în loc de atribuire, returnare și/sau dare ca parametru de structuri se atribuie/returnează/dau ca parametri **pointeri la structuri**.

⇒ acces la câmpuri prin intermediul unor pointeri la structuri

Operatorul -> e echivalent cu **indirectarea** urmată de **selecție**:

pointer -> nume-camp \iff **(*pointer).nume-camp**

```
struct student s, *p;  
p = &s;  
(*p).nota_dipl = 9.50; //echivalent cu  
p -> nota_dipl = 9.50;
```

Obs: Operatorii . și -> au precedența **cea mai ridicată**, ca () și []!

Tablouri și structuri

În C putem avea tablouri de structuri sau structuri care conțin tablouri (tipurile **agregat** pot fi combinate arbitrar).
⇒ se alege varianta care grupează logic datele

```
char* nluni[12] = { "ian", /*...,*/ "dec" };
char zile_luna[12]
    = { 31, 28, 31, 30, /*...,*/ 30, 31 };

// e preferabila varianta urmatoare
struct luna {
    char *nume;
    int zile;
};

struct luna luni[12]
= {{"ian",31}, /*...,*/ {"dec",31}};
```

Structuri de date recursive

Un câmp al unei structuri nu poate fi o structură de același tip (rezultă o structură de dimensiune infinită!).

Dar poate fi **adresa unei structuri de același tip**
⇒ **structuri de date recursive**

Exemplu – o listă de cuvinte:

```
// struct wl e un tip , incomplet definit
struct wl {
    char *word; // cuvantul
    struct wl *next;
    // pointer la structura de acelasi tip
};
// definiri\c tia e complet\u a
```

Structuri recursive – exemple:

Un arbore binar, având informația din noduri numere întregi:

```
typedef struct t tree;
// defineste tipul incomplet tree = struct t

struct t {
    int val;
    tree *left, *right;
    // foloseste numele din typedef
};
// aici tipul struct t e complet
// si echivalent cu tree
```

Structuri cu câmpuri pe biți

Structuri cu câmpuri pe biți se folosesc pentru reprezentarea mai compactă a informației, ocupând doar atât spatiu cat e nevoie.

```
struct date_t {  
    unsigned sec, min : 6; // nr. de biti  
    // se permit tipuri intregi  
  
    unsigned hour, day: 5;  
    unsigned month: 4;  
    unsigned year: 6;  
} data = {0, 0, 17, 19, 5, 39 };  
  
//...  
printf("%u.%u\n", data.day, data.month);
```

Tipul uniune

Tipul uniune – pt. a retine valori care pot avea **tipuri diferite**.

Sintaxa: ca la structuri, dar cu cuvântul cheie **union** în față.

Diferența dintre **struct** și **union** – pt. **union** lista de câmpuri reprezinta o **listă de variante**, pentru fiecare tip:

- o variabilă **structură** conține **toate** câmpurile declarate
- o variabilă **uniune** conține **exact una** din variantele declarate.

Dimensiunea unui tip uniune este dată de cel mai mare tip din lista de variante.

Tipul uniune – exemplu

```
union u{ // tip uniune
    int i;
    double r;
    char *s;
} val; // trei variante pentru fiecare tip

enum { INT, REAL, SIR } tip;
// tine minte varianta memorata

val.r = 3.14; tip = REAL;
// sau
val.i = 100; tip = INT;
// sau
val.s = "ceva"; tip = SIR;
```