

Generalități: Tipul enumerare

Un tip definește o *mulțime de valori* și *operațiile* posibile cu acestea. În C putem defini tipuri *enumerare*, *structură* și *unione*.

Tipul enumerare: dă nume unui șir de valori numerice.

⇒ Folosit când e mai sugestiv de scris un nume decât un număr

```
enum luni_curs {ian=1, feb, mar, apr, mai, iun, iul, oct=10, nov, dec};
```

definește un tip cu numele `enum luni_curs` (și `enum` e *parte din numele*)

Implicit, șirul valorilor e crescător începând cu 0

dar putem specifica și explicit valori (și o valoare se poate repeta)

Un tip enumerare e un tip întreg

⇒ variabilele/valorile enumerare se folosesc ca și întregi

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

Un nume de constantă nu poate fi folosit în mai multe enumerări

Programarea calculatoarelor. Curs 11

Marius Minea

Programarea calculatoarelor**Tipuri definite de utilizator**

19 mai 2009

Programarea calculatoarelor. Curs 11

Marius Minea

Tipuri definite de utilizator

Tipuri structură

3

Grupează mai multe elemente de tipuri diferite, legate logic între ele

```
struct student { // numele complet al tipului (incl. "struct")
    char nume[32], prenume [32]; // două tablouri de caractere
    char *domiciliu; // ADRESA! memoria pt. șir se alocă altundeva
    char nr_tel[10]; // max. 9 cifre + terminator \0
    float medie_an[4]; // declaratiile campurilor de structura
    float nota_dipl; // arata la fel ca declaratiile de variabile
    } stud1, stud2; // două variabile declarate direct aici
    struct student s3; // declarația separată de variabila
```

Elementele unei structuri se numesc *câmpuri* (engl. fields)

pot fi de orice tip, dar **NU** de *acelasi* tip structură (nu recursiv)

Numele câmpurilor se văd doar în interiorul structurii

⇒ tipuri structurii diferite pot avea câmpuri numite la fel

Programarea calculatoarelor. Curs 11

Marius Minea

Tipuri definite de utilizator

Folosirea structurilor (cont.)

5

Structurile *pot* fi atribuite în totalitatea lor.

```
struct point p1, p2; p1 = p2;
```

Structurile *pot* fi transmise către / returnate de funcții.

Pt. dimensiuni mari, se preferă transmiterea / returnarea de pointeri.

Structurile *nu pot* fi comparate cu operatori logici

⇒ trebuie comparate individual câmpurile lor

Declararea de tipuri: `typedef nume-tip-existent nume-tip-nou;`

ex. `typedef double real;`

⇒ putem alege un nume mai scurt, fără struct.

— direct în definiția tipului:

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

— sau separat de definiția tipului structură propriu-zis:

```
typedef struct student { /* ceva campuri */ }; // definește tipul
typedef struct student_t; // declară un nume pentru el
```

Programarea calculatoarelor. Curs 11

Marius Minea

Tipuri definite de utilizator

Folosirea structurilor. Operatori

4

1) toată structura (atribuite, date ca parametru, returnate ca rezultat)

2) accesul la câmpuri: se face cu sintaxa *nume-variabila.nume-câmp*

Punctul `.` e **operatorul de selecție** (e un operator postfix)

```
struct student s; // var. s: numele complet al tipului: struct student
strcpy(s.nume, "Stefanovici"); // NU! s.nume = ... (e un tablou)
s.domiciliu = "str. Iinstei nr. 2"; // sau malloc + strcpy
s.medie_an[2] = 9.35; // un câmp se folosește ca orice variabilă
```

Inițializarea structurilor: câmp cu câmp, cu acolade, ca la tablouri

```
struct point { float x, y; } pct1 = { 2.5, 1.5 };
```

Putem scrie direct valori de tip structură (**literate compuse**)

(*nume-tip*) { *inicializatori* } // tip indicat explicit în conversie

```
void draw(struct point p); // declarație de funcție
struct point p2; // declară o variabilă structură
p2 = (struct point) { -1, 2 }; // indică ce tip are valoarea dată
draw((struct point) { 1.5, 2.5 }); // struct. dată ca parametru
```

Programarea calculatoarelor. Curs 11

Marius Minea

Tipuri definite de utilizator

Pointeri la structuri

6

Frevent: accesul la câmpuri prin intermediul unui pointer la structură:

```
struct student *p; /* p = ... */ (*p).nota_dipl = 9.50;
```

Operatorul `->` e echivalent cu îndreptarea următa de selecție:

```
pointer->nume_câmp e echivalent cu (*pointer).nume_câmp
```

Operatorii `.` și `->` au precedența cea mai ridicată, ca și `()` și `[]`

Atenție la ordinea de evaluare !

```
p->x++ inseamnă (p->x)++
++p->x inseamnă ++(p->x)
*p->x inseamnă *(p->x)
*p->s++ inseamnă *((p->s)++)
```

Programarea calculatoarelor. Curs 11

Marius Minea

Tipuri definite de utilizator

Structuri și tablouri

7

În C, tipurile agregat pot fi combinate arbitrar (tablouri de structuri, structuri cu câmpuri de tip tablou, etc.)

Tipurile trebuie definite în așa fel încât să grupeze logic datele.

Ex.: dacă două tablouri au același domeniu pt. indici și datele de la același indice sunt folosite împreună, e preferabilă gruparea în structură:

```
char* nume_luna[12] = {"ianuarie", /* ... , */ "decembrie"};
char zile_luna[12] = { 31, 28, 31, 30, /* ... , */ 30, 31 };
// e preferabilă varianta următoare

struct luna {
    char *nume;
    int zile;
};

struct luna luna[12] = {"ianuarie",31}, /*...*/ {"decembrie",31};
```

Programarea calculatoarelor. Curs 11

Marius Minea

Tipuri definite de utilizator

Structuri cu câmpuri pe biți

9

Vrem să reprezentăm mai multe informații cât mai compact pe biți.

Ex.: o dată ca întreg pe 32 de biți: sec, min (0-59): 6 biți, ora (0-23), ziua (1-31): 5 biți, luna (1-12): 4 biți), anul (1970 + 0-63): 6 biți.

Constuim : int data = 39 << 26 | 5 << 22 | 19 << 17 | 17 << 12; (19.5.2009, 17h). Extragem ora: int ora = data >> 12 & 0x1F;

Sau: acces direct la câmpuri pe biți, fără măști și operatori pe biți.

```
struct date_t { // alternativa: structură cu câmpuri pe biți
    unsigned sec, min : 6; // indică numărul de biți
    unsigned hour, day: 5; // se permit tipuri întregi
    unsigned month: 4;
    unsigned year: 6;
    } data = {0, 0, 17, 19, 5, 39 }; // 17:00:00, 19.05.(1970+39)

Putem scrie direct: printf("%u.%u.\n", data.day, data.month);
Putem avea câmpuri fără nume: int: 2; // pe 2 biți
sau forța trecerea la memorarea în octetul următor int: 0;
```

Programarea calculatoarelor. Curs 11

Marius Minea

Tipuri definite de utilizator

11

Programe compuse din mai multe fișiere

Programele se scriu modular, într-un fișier .c se pun funcții înrudite (exemplu: un fișier *biblioteca* de funcții pentru lucrul cu un anumit tip)

Declarațiile de tipuri, funcții și variabile care trebuie folosite și în alte fișiere se pun într-un fișier antet .h

Acosta e inclus de fiecare fișier .c care îl necesită.

Fișierul biblioteca.c poate fi *compilat separat* (gcc -o) într-un *fișier obiect* .o – acesta conține cod masină, dar cu informații despre numele de funcții/variabile, pentru a putea fi folosite

Programatorul scrie un program main.c care include biblioteca.h și e compilat împreună cu codul bibliotecii (e suficient fișierul obiect).

Programarea calculatoarelor. Curs 11

Marius Minea

Tipuri definite de utilizator

Structuri de date recursive

8

Un câmp al unei structuri nu poate fi o structură de același tip (s-ar obține o structură de dimensiune infinită/nedefinită).

Poate fi însă *adresa* unei structuri de același tip (un pointer)!
⇒ structuri de date recursive, înălțute (liste, arbori, etc.)

```
struct wl { // struct wl e un tip, incomplet definit
    char *word; // cuvântul: informația propriu-zisă
    struct wl *next; // pointer la structura de același tip
}; // acum definiția tipului e completă
```

Un arbore binar, având în noduri numere întregi:

```
typedef struct t tree; // definește tipul incomplet tree = struct t
struct t {
    int val;
    tree *left, *right; // folosește numele din typedef
}; // aici tipul struct t e complet și echivalent cu tree
```

Programarea calculatoarelor. Curs 11

Marius Minea

Tipuri definite de utilizator

Uniuni

10

Folosite pentru a reține valori care pot avea tipuri diferite.

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

Lista de câmpuri reprezintă o listă de variante, pentru fiecare tip:

– o variabilă structură conține toate câmpurile declarate

– o variabilă uniune conține exact *una* din variantele date

```
union { // tip uniune, fara nume
    int i;
    double r;
    char *s;
} val; // trei variante pentru fiecare tip de valoare
enum { INT, REAL, SIR } tip; // tine mână varianta memorata
char s[32]; if (scanf("%31s", s) == 1) {
    if (isdigit(*s)) // incepe cu cifra ? daca da, contine punct ?
        if (strchr(s, '.') ) { sscanf(s, "%lf", &val.r); tip = REAL; }
        else { sscanf(s, "%d", &val.i); tip = INT; }
    } else { val.s = strdup(s); tip = SIR; }
}
```

Programarea calculatoarelor. Curs 11

Marius Minea

Tipuri definite de utilizator

Tipuri de date abstracte

12

TDA = un model matematic cu un set de operații asupra lui
⇒ o structură de date + funcții care operează pe ea

⇒ noțiunea de *clasă* din programarea orientată pe obiecte

Pentru implementarea TDA în C:

– în fișierul .h se declară minimul necesar pentru a putea compila programul (pentru structuri, adesea doar un typedef pt. pointer la tip) – și declarații de funcții care manipulează tipul respectiv

– structura tipului și definițiile funcțiilor: ascunse în implementare (.c)

```
typedef struct node *list_t; /* în fișierul .h */
typedef struct node { /* în fișierul .c cu implementarea */
    int info; /* sau/și alte câmpuri */
    struct node *nxt;
} node_t; /* tip vizabil doar în fișierul .c */
```

Utilizatorul, care include doar fișierul .h nu are acces la structura internă a tipului (node_t); accesul e permis doar prin funcții.

⇒ schimbări în implementarea bibliotecii nu afectează programul.

Programarea calculatoarelor. Curs 11

Marius Minea