Capitolul IB.09. Structuri de date. Definire şi utilizare în limbajul C

Cuvinte cheie
Structura(struct), câmp, typedef, structuri predefinite, structuri cu conținut variabil (union), enumerări

IB.09.1. Definirea de tipuri şi variabile structură

O structură este o colecție de valori eterogene ca tip, stocate într-o zonă compactă de memorie.

Cu alte cuvinte, o structură este un tip de date care permite gruparea unor date de tipuri diferite sub un singur nume. Componentele unei structuri, denumite **câmpuri**, sunt identificate prin nume simbolice, denumite **selectori**. Câmpurile unei structuri pot fi de orice tip, simplu sau derivat, dar nu *void* sau funcție. Printr-o declarație *struct* se defineşte un nou tip de date de tip structură, de către programator.

IB.09.1.1 Declararea structurilor

Declararea structurilor se face folosind cuvântul cheie ***struct***; definirea unui tip structură are sintaxa următoare:

Sintaxă:

```
struct nume_structură {
        tip_câmp1 nume_câmp1;
        tip_câmp2 nume_câmp2;
        …
} [lista_variabile_structură];
```

unde:

- ***nume_structura*** este un nume de tip folosit numai precedat de cuvântul cheie *struct* (în C, dar în C++ se poate folosi singur ca nume de tip).
- ***tip_câmp1, tip_câmp2***,... este tipul componentei (câmpului) i
- ***nume_câmp1, nume_câmp2***,... este numele unei componente (câmp)

Exemple:

```
//structura numar complex
struct Complex {
  double real;
  double imag;
};
```

Ordinea enumerării câmpurilor unei structuri nu este importantă, deoarece ne referim la câmpuri prin numele lor. Se poate folosi o singura declarație de tip pentru mai multe câmpuri:

```
//structura moment de timp
struct time {
   int ora, min, sec;
};

//structura activitate
struct activ {
   char numeact[30];          // nume activitate
   struct time start;         // ora de incepere
   struct time stop;          // ora de terminare
};
```

Nume_structura sau **lista_variabile_structura** pot lipsi, dar nu simultan. Dacă se precizează **nume_structura**, atunci înseamnă că se face definirea tipului **struct nume_structura**, care poate fi apoi folosit pentru declararea de variabile, ca tip de parametri formali sau ca tip de rezultat returnat de funcţii.
Declararea unor variabile de un tip structură se poate face fie după declararea tipului structură, fie simultan cu declararea tipului structură.
Nu există constante de tip structură, dar este posibilă iniţializarea la declarare a unor variabile structură. Astfel de variabile iniţializate şi cu atributul *const* ar putea fi folosite drept constante simbolice.

Exemple:

```
struct time t1,t2, t[100];   //t este vector de structuri

struct complex {
        float re,im;
} c1, c2, c3;                 //numere complexe

struct complex cv[200];       //un vector de numere complexe

struct coordonate{      //se declara tipul struct coordonate
    float x,y;
} punct, *ppunct, puncte[20]; //variabile
struct coordonate alt_punct = {1,4.5},
                  alte_puncte[10] = {{2,4},{8},{9,7}},
                  mai_multe_puncte[25] = {1,2,3,4};
//variabilele de tip structura se pot initializa la declarare;
//campurile neprecizate sunt implicit 0


struct persoana{             //se declara tipul struct persoana
     char nume[20];
     int varsta;
};                          //lipseste lista_declaratori
struct persoana pers={"Ion Ionescu",21}, *ppers, persoane[12];

struct{                     //lipseste nume_struct
     char titlu[20], autor[15], editura[12];
     int an_aparitie;
 }carte, biblioteca[1000];  /* nu se declara un tip, deci doar aici pot fi
                            facute declaratiile de variabile */
```

Un câmp al unei structuri poate fi de tip structură, dar nu aceeaşi cu cea definită - se poate însă să se declare un câmp pointer la structura definită (aspect care va fi utilizat la implementarea listelor):

```
struct persoana{                //se declara tipul struct persoana
     char nume[20];
     struct{
       int zi,an,luna

    }data_nasterii;             //camp de tip structura
 }p;
struct nod_lista{
    tip_info info;
    struct nod_lista * urm;    //camp pointer la structura definita
};
```

Numele de structuri se află într-un spațiu de nume diferit de cel al numelor de variabile - se pot declara deci variabile şi tipuri structură cu acelaşi nume - de evitat însă. Se pot declara tipuri structuri care au nume de câmpuri identice.

De remarcat că orice declarație *struct* se termină obligatoriu cu caracterul ';' chiar dacă acest caracter urmează după o acoladă; aici acoladele nu delimitează un bloc de instrucțiuni ci fac parte din declarația *struct.*
În structuri diferite pot exista câmpuri cu acelaşi nume, dar într-o aceeaşi structură numele de câmpuri trebuie să fie diferite.

Accesul la câmpurile unei variabile de tip structură se face utilizând operatorul punct (.).
Exemplu:

```
struct complex c1;
...
c1.re

struct time t2;
...
t2.ora

struct time t[10];
...
t[0].min.
```

Atenție! Câmpurile unei variabile structură nu se pot folosi decât dacă numele câmpului este precedat de numele variabilei structură din care face parte, deoarece există un câmp cu acelaşi nume în toate variabilele de un acelaşi tip structură.
Exemplu:

```
int main () {
  complex c1,c2;
  scanf ("%f%f", &c1.re, &c1.im);            // citire c1
  c2.re= c1.re; c2.im= -c1.im;               // complex conjugat
  printf ("(%f,%f) ", c2.re, c2.im);         // scrie c2
}
```

Dacă un câmp este la rândul lui o structură, atunci numele câmpului poate conține mai multe puncte ce separă numele variabilei şi câmpurilor de care aparține (în ordine ierarhică).
Exemplu:

```
//structura moment de timp
struct time {
   int ora, min, sec;
};

//structura activitate
struct activ {
   char numeact[30];          // nume activitate
   struct time start;         // ora de incepere
   struct time stop;          // ora de terminare
};
....
struct activ a;
printf ("%s începe la %d: %d şi se termina la %d: %d \n", a.numeact,
a.start.ora, a.start.min, a.stop.ora, a.stop.min);
```

IB.09.2. Asocierea de nume sinonime pentru tipuri structuri - typedef

Printr-o declarație *struct* se definește un nou tip de date de către programator. Utilizarea tipurilor structură pare diferită de utilizarea tipurilor predefinite, prin existența a două cuvinte care desemnează tipul (*struct* numestr). Declarația ***typedef*** din C permite atribuirea unui nume oricărui tip, nume care se poate folosi apoi la fel cu numele tipurilor predefinite ale limbajului. Sintaxa declarației *typedef* este la fel cu sintaxa unei declarații de variabilă, dar se declară un nume de tip şi nu un nume de variabilă.
În limbajul C declarația *typedef* se utilizează frecvent pentru atribuirea de nume unor tipuri structură.
Exemple:

```
// definire nume tip simultan cu definire tip structură
typedef struct {
   float re,im;
} complex;

// definire nume tip după definire tip structura
typedef struct activ act;
```

Deoarece un tip structură este folosit în mai multe funcții (inclusiv *main*), definirea tipului structură (cu sau fără *typedef*) se face la începutul fişierului sursă care conține funcțiile (înaintea primei funcții). Dacă un program este format din mai multe fişiere sursă atunci definiția structurii face parte dintr-un fişier antet (de tip ***.h***).

Se pot folosi ambele nume ale unui tip structură (cel precedat de *struct* şi cel dat prin *typedef*), care pot fi chiar identice.
Exemple:

```
typedef struct complex  {
   float re;
   float im;
} complex;

typedef struct point {
   double x,y;
} point;
...
struct point p[100];

// calcul arie triunghi dat prin coordonatele varfurilor
double arie ( point a, point b, point c) {
   return  a.x * (b.y-c.y) - b.x * (a.y-c.y) + c.x * (a.y-b.y);
}

typedef struct card
{
   int val;
   char cul[20];
} Carte;
....
Carte c1, c2;
c1. val = 10;
strcpy (c1. cul, "caro");
c2 = c1;
```

Cu *typedef* structura poate fi şi anonimă (poate lipsi cuvantul card din ultimul exemplu):

```
typedef struct
{
   int val;
   char cul[20];
} Carte;
```

Atunci când numele unui tip structură este folosit frecvent, inclusiv în parametri de funcții, este preferabil un nume introdus prin *typedef*, dar dacă vrem să punem în evidență că este vorba de tipuri structură vom folosi numele precedat de cuvântul cheie *struct*.
În C++ se admite folosirea numelui unui tip structură, fără a fi precedat de *struct* şi fără a mai fi necesar *typedef*.

IB.09.3. Utilizarea tipurilor structură

Un tip structură poate fi folosit în:
- declararea de variabile structuri sau pointeri la structuri
- declararea unor parametri formali de funcții (structuri sau pointeri la structuri)
- declararea unor funcții cu rezultat de un tip structură

Operațiile posibile cu variabile de un tip structură sunt:
- Selectarea unui câmp al unei variabile structură se realizează folosind operatorul de selecție **.** . Cîmpul selectat se comportă ca o variabilă de acelaşi tip cu câmpul, deci i se pot aplica aceleaşi prelucrări ca oricărei variabile de tipul respectiv.

variabila_structura.nume_camp

Exemplu:

```
struct persoana{              //se declara tipul struct persoana
    char nume[20];
    struct {
      int zi,an,luna
    } data_nasterii;          //camp de tip structura
} p;


//Selecția câmpurilor pentru variabila p de mai sus:

p.nume                                  //tablou de caractere
p.nume[0]                               //primul    caracter    din    nume
p.nume[strlen(p.nume)-1]                 //ultimul    caracter    din    nume
p.data_nasterii.an
p.data_nasterii.luna
p.data_nasterii.an
```

- O variabilă structură poate fi inițializată la declarare prin precizarea între {} a valorilor câmpurilor; cele neprecizate sunt implicit 0.

- O variabilă structură poate fi copiată în altă variabilă de acelaşi tip.
  Exemplu cu declarațiile de mai sus:

```
printf("%d %d\n", sizeof(pers), sizeof(struct persoana));
ppers = &pers;
persoane[0] = *ppers;      //atribuirea intre doua variabile structura
```

- O variabilă structură nu poate fi citită sau scrisă direct, ci prin intermediul câmpurilor!!
  Se pot aplica operatorii:
  & - referință
  sizeof - dimensiune

- **->**. Deoarece structurile se prelucrează frecvent prin intermediul pointerilor, a fost introdus operatorul -> care combină operatorul de indirectare '*' cu cel de selectie'**.'**. Cele două expresii de mai jos sunt echivalente:

(*variabila_pointer).nume_camp ⇔ variabila_pointer->nume_camp

- Transmiterea ca argument efectiv la apelarea unei funcții;
- Transmiterea ca rezultat al unei funcții, într-o instrucțiune *return.*

Singurul operator al limbajului C care admite operanzi de tip structura este cel de atribuire. Pentru alte operații cu structuri trebuie definite funcții: comparații, operații aritmetice, operații de citire-scriere etc.
Exemplul următor arată cum se poate ordona un vector de structuri time:

```
struct time {                    //structura moment de timp
   int ora, min, sec;
};

void wrtime ( struct time t) {// scrie ora, min, sec
  printf ("%02d:%02d:%02d \n", t.ora,t.min,t.sec);
}

int cmptime (struct time t1, struct time t2) {      // comparare
  int d;
  d=t1.ora - t2.ora;
  if (d) return d;
  d=t1.min - t2.min;               // <0 daca t1<t2 şi >0 daca t1>t2
  if (d) return d;                 // rezultat negativ sau pozitiv
  return t1.sec - t2.sec;          // rezultat <0 sau =0 sau > 0
}

// utilizare funcții
int main () {
  struct time tab[200], aux;
  int i, j, n;
  . . .    // citire date
  // ordonare vector
  for (j=1;j<n;j++)
    for (i=1;i<n;i++)
      if ( cmptime (tab[i-1],tab[i]) > 0) {
        aux=tab[i-1];
        tab[i-1]=tab[i];
        tab[i]=aux;
    }
  // afişare vector ordonat
  for (i=0;i<n;i++)
    wrtime(tab[i]);
}
```

Principalele avantaje ale utilizării unor tipuri structură sunt:

- Programele devin mai explicite dacă se folosesc structuri în locul unor variabile separate.
- Se pot defini tipuri de date specifice aplicației iar programul reflectă mai bine universul aplicației.
- Se poate reduce numărul de parametri al unor funcții prin gruparea lor în parametri de tipuri structură şi deci se simplifică utilizarea acelor funcții.
- Se pot utiliza structuri de date extensibile, formate din variabile structură alocate dinamic şi legate între ele prin pointeri (liste înlănțuite, arbori ş.a).

IB.09.4. Funcţii cu parametri şi/sau rezultat structură

O funcţie care produce un rezultat de tip structură poate fi scrisă în două moduri, care implică şi utilizări diferite ale funcţiei:

- funcţia are rezultat de tip structură:

```
// citire numar complex (varianta 1)
complex readx () {
   complex c;
   scanf ("%f%f",&c.re, &c.im);
   return c;
}

// utilizare
complex a[100];
for (i=0;i<n;i++)
   a[i]=readx();
```

- funcţia este de tip *void* şi depune rezultatul la adresa primită ca parametru (pointer la tip structură):

```
// citire numar complex (varianta 2)
void readx ( complex * px) {       // px pointer la o structură complex
   scanf ("%f%f", &px->re, &px->im);
}

// utilizare
complex a[100];
......
for (i=0;i<n;i++)
   readx (&a[i]);                  // adresa variabilei structură a[i]
```

Reamintim că notaţia px->re este echivalentă cu notaţia (*px).re şi se interpretează astfel: "câmpul *re* al structurii de la adresa *px*".

Uneori, mai multe variabile descriu împreună un anumit obiect şi trebuie transmise la funcţiile ce lucrează cu obiecte de tipul respectiv. Gruparea acestor variabile într-o structură va reduce numărul de parametri şi va simplifica apelarea funcţiilor. Exemple de obiecte definite prin mai multe variabile: obiecte geometrice (puncte, poligoane ş.a), date calendaristice şi momente de timp, structuri de date (stiva, coada, ş.a), vectori, matrice, etc.

Exemplu de grupare într-o structură a adresei şi dimensiunii unui vector:

```
typedef struct {
   int vec[1000];
   int dim;
} vector;

// afişare vector
void scrvec (vector v) {
   int i;
   for (i=0;i<v.dim;i++)
      printf ("%d  ",v.vec[i]);
   printf ("\n");
}

// elementele comune din 2 vectori
vector comun (vector a, vector b) {
   vector c;
   int i,j,k=0;
   for (i=0;i<a.dim;i++)
      for (j=0;j<b.dim;j++)
```

```
        if (a.vec[i]==b.vec[j])
          c.vec[k++]=a.vec[i];
   c.dim=k;
   return c;
}
```

Pentru structurile care ocupă un număr mare de octeți este mai eficient să se transmită ca parametru la funcții adresa structurii (un pointer) în loc să se copieze conținutul structurii la fiecare apel de funcție şi să se ocupe loc în stivă, chiar dacă funcția nu face nici o modificare în structura a cărei adresă o primeşte.
Funcțiile cu parametri pointeri la structuri pot produce efecte secundare (laterale) nedorite, prin modificarea involuntară a unor variabile din alte funcții.

În concluzie, avantajele utilizării tipurilor structură sunt următoarele:

- Programele devin mai explicite dacă se folosesc structuri în locul unor variabile separate.
- Se pot defini tipuri de date specifice aplicației iar programul reflectă mai bine universul aplicației.
- Se poate reduce numărul de parametri al unor funcții prin gruparea lor în parametri de tipuri structură şi deci se simplifică utilizarea acelor funcții.
- Se pot utiliza structuri de date extensibile, formate din variabile structură alocate dinamic şi legate între ele prin pointeri (liste înlănțuite, arbori s.a).

Exemple
1.Să se definească o structură Point pentru un punct geometric 2D şi o structură Rectangle pentru un dreptunghi definit prin colțul stânga sus şi colțul dreapta jos. Să se inițializeze şi să se afişeze o variabilă de tip Rectangle.

```
#include <stdio.h>

typedef struct Point {
  int x, y;
} Point;

typedef struct Rectangle {
  Point topLeft;
  Point bottomRight;
} Rectangle;

int main() {
 Point p1, p2;
 p1.x = 0;  // p1 la (0, 3)
 p1.y = 3;
 p2.x = 4;  // p2 la (4, 0)
 p2.y = 0;
 printf( "p1 la ( %d, %d)\n", p1.x, p1.y);
 printf( "p2 la ( %d, %d)\n", p2.x, p2.y);

 Rectangle rect;
 rect.topLeft = p1;
 rect.bottomRight = p2;
 printf("Stanga sus la(%d,%d)\n", rect.topLeft.x,rect.topLeft.y);
 printf("Dreapta jos la (%d,%d)\n", rect.bottomRight.x,rect.bottomRight.y);
 return 0;
}
```

Rezultatul va fi:

```
p1 la (0,3)
p2 la (4,0)
Stanga sus la (0,3)
Dreapta jos la (4,0)
```

2. Să se defineasca o structura “time" care grupează 3 întregi ce reprezintă ora, minutul şi secunda pentru un moment de timp. Să se scrie funcţii pentru:

- Verificare corectitudine ora
- Citire moment de timp
- Scriere moment de timp
- Comparare de structuri “time".

Program pentru citirea şi ordonarea cronologică a unor momente de timp şi afişarea listei ordonate, folosind funcţiile anterioare.

```
#include<stdio.h>

typedef struct time {
    int ora, min, sec;
}time;

void wrtime ( time t) {       // scrie ora, min, sec
      printf ("%02d:%02d:%02d \n", t.ora,t.min,t.sec);
}

int cmptime (time t1, time t2) {    // compara momente de timp
  int d;

  d=t1.ora - t2.ora;
  if (d) return d;
  d=t1.min - t2.min;                // <0 daca t1<t2 şi >0 daca t1>t2
  if (d) return d;                  // rezultat negativ sau pozitiv
  return t1.sec - t2.sec;           // rezultat <0 sau =0 sau > 0
}

int corect (time t) {               // verifica daca timp plauzibil
      if ( t.ora < 0 || t.ora > 23 ) return 0;
  if ( t.min < 0 || t.min > 59 ) return 0;
  if ( t.sec < 0 || t.sec > 59 ) return 0;
  return 1;                         // plauzibil corect
}

time rdtime () {                    // citire ora
  time t;

  do {
            scanf ("%d%d%d", &t.ora, &t.min,&t.sec);
            if ( ! corect (t))
              printf ("Date gresite, repetati introducerea: \n");
            else break;
  }while(1);

  return t;
}

void sort (time a[], int n) {      // ordonare vector de date
      int i,gata;
```

```
       time aux;

      do {
         gata=1;
         for (i=0;i<n-1;i++)
             if (cmptime(a[i],a[i+1]) > 0 ) {
                  aux=a[i];
          a[i]=a[i+1];
          a[i+1]=aux;
          gata=0;
                  }
       }while (!gata);

}

int main () {
        time t[30];
        int n,i;
        printf("introducere n: ");
       scanf("%d",&n);
        for(i=0;i<n;i++) t[i] = rdtime();
        sort (t, n);
        for ( i=0 ;i<n ;i++) wrtime(t[i]);
        return 0;
}
```

IB.09.5. Structuri predefinite

În bibliotecile C standard există unele structuri predefinite. Un astfel de exemplu este structura ***struct tm*** definită în biblioteca ***time***; această structură conține componente ce definesc complet un moment de timp:

```
struct tm {
  int    tm_sec, tm_min, tm_hour;        // secunda, minut, ora
  int    tm_mday,  tm_mon,  tm_year;     // zi, luna, an
  int    tm_wday, tm_yday;               // nr zi în saptamana şi în an
  int    tm_isdst;                       // 1 - se modifica ora (iarna/vara), 0 - nu
};
```

Există funcții care lucrează cu această structură: ***asctime***, ***localtime***, etc.

Exemplu: cum se poate afişa ora şi ziua curentă, folosind numai funcții standard

```
#include <stdio.h>
#include <time.h>
int main(void) {
  time_t t;                   // time_t  este alt nume pentru  long
  struct tm *area;            // pentru rezultat funcție localtime
  t = time (NULL);            // obtine ora curenta
  area = localtime(&t);       // conversie din time_t în struct tm
  printf ("Local time is: %s", asctime(area));
}
```

Observații:

- ***asctime(struct tm* t)*** returnează un şir ce reprezintă ziua şi ora din strucura t. Şirul are următorul format: DDD MMM dd hh:mm:ss YYYY
- funcția ***time*** transmite rezultatul şi prin numele funcției şi prin parametru: ***long time (long*)***, deci se putea apela şi: ***time (&t);***

O altă structură predefinită este ***struct stat*** definită în fişierul ***sys/stat.h. Structura*** reuneşte date despre un fişier, cu excepția numelui:

```
struct stat {
  short unix [7];              // fără semnificatie în sisteme Windows
  long st_size;                // dimensiune fisier (octeti)
  long st_atime, st_mtime;     // ultimul acces / ultima modificare
  long st_ctime;               // data de creare
};
```

Funcția ***stat*** completează o astfel de structură pentru un fişier cu nume dat:

```
int stat (char* filename,  struct stat * p);
```

Exemplu:
Pentru a afla dimensiunea unui fişier normal (care nu este fişier director) vom putea folosi funcția următoare:

```
long filesize (char * filename) {
   struct stat fileattr;
   if (stat (filename, &fileattr) < 0)          // daca fisier negasit
      return -1;
   else     // fisier gasit
      return (fileattr.st_size);     // campul st_size contine lungimea
}
```

IB.09.6. Structuri cu conținut variabil (uniuni)

Uniunea (reuniunea – union) defineşte un grup de variabile care nu se memorează simultan ci alternativ.
În felul acesta se pot memora diverse tipuri de date la o aceeaşi adresă de memorie.
Cuvântul cheie ***union*** se foloseşte la fel cu *struct*. Alocarea de memorie se face (de către compilator) în funcție de variabila ce necesită maxim de memorie.

Declararea uniunilor se face folosind cuvântul cheie ***union:***

Sintaxă:

```
union {
        tip_comp1 nume_comp1;
        tip_comp2 nume_comp2;
        …
} [lista_variabile_structură];
```

unde:

- ***tip_comp1, tip_comp2***,... este tipul componentei i
- ***nume_comp1, nume_comp2***,... este numele unei componente

Exemplu:

```
union {
     int ival;
    long lval;
    float fval;
    double dval;
} val;
```

O uniune face parte de obicei dintr-o structură care mai conține şi un *câmp discriminant*, care specifică tipul datelor memorate (alternativa selectată la un moment dat).

Exemplul următor arată cum se poate lucra cu numere de diferite tipuri şi lungimi, reunite într-un tip generic:

```
typedef struct numar {
  char tipn;      // tip numar (caracter: i-int, l-long, f-float, d-double
  union {
      int ival;
     long lval;
     float fval;
     double dval;
  } val;                      // valoare numar
} Numar;                      // definire tip de date Numar

void write (Numar n) {        // in functie de tip afiseaza valoarea
  switch (n.tipn) {
   case 'i': printf ("%d ", n.val.ival); break;
   case 'l': printf ("%ld ", n.val.lval); break;
   case 'f': printf ("%f ", n.val.fval); break;
   case 'd': printf ("%.15lf ", n.val.dval);
  }
}
```

Observaţie:
În locul construcţiei *union* se poate folosi o variabilă de tip *void** care va conţine adresa unui număr, indiferent de tipul lui. Memoria pentru număr se va aloca dinamic:

```
typedef struct number {
   char tipn;           // tip numar
   void * pv;           // adresa numar
}number;

// in functie de tip afiseaza valoarea
void write (number n) {
   switch (n.tipn) {
            case 'i': printf("%d",*(int*)n.pv);
                      //conversie la int* si indirectare
               break;
      ...
      case 'd': printf("%.15lf",*(double*)n.pv); /*conversie la double *
                                                   si indirectare*/
               break;
  }
}
```

IB.09.7. Enumerări

Tipul enumerare este un caz particular al tipurilor întregi. Se utilizează pentru a realiza o reprezentare comodă şi sugestivă a unor obiecte ale caror valori sunt identificate printr-un număr finit de nume simbolice.
Tipul enumerare declară **constante simbolice**, cărora li se **asociază coduri numerice de tip întreg**. Compilatorul asociaza constantelor enumerate câte un cod întreg din succesiunea începând cu 0. Implicit, şirul valorilor e crescător cu pasul 1.
Un nume de constantă nu poate fi folosit în mai multe enumerări.

Sintaxa este următoarea:

Sintaxă:
enum [nume_tip] { lista_constante } [lista_variabile] ;

unde *nume_tip* şi *lista de variabile* pot lipsi!

Exemple:

```
enum zile_lucr { luni, marti, miercuri, joi, vineri };
// luni e asociat cu 0, marti cu 1, ..., vineri cu 4

// se pot defini variabile de tipul zile_lucr, ca mai jos, variabila zi:
enum zile_lucr zi=marti;

enum Color {
   red, green, blue
} myColor;      // Defineste o enumerare şi declară o variabilă de tipul ei
......
myColor = red;          // Atribuie o valoare variabilei
Color yourColor;        // Declară o variabilă de tipul Color
yourColor = green;      // Atribuie o valoare variabilei
```

Dacă se doreşte o altă codificare a constantelor din enumerare decât cea implicită, pot fi folosite în enumerare elemente de forma:

nume_constantă = valoare_întreagă;

Constantelor simbolice ce urmează unei astfel de iniţializări li se asociază numerele întregi următoare:

```
enum transport { tren, autocar=5, autoturism, avion };
/* tren e asociat cu 0, autocar cu 5, autoturism cu 6, avion cu 7 */

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

După cum spuneam, tipurile enumerare sunt tipuri întregi, variabilele enumerare se pot folosi la fel ca variabilele întregi, asigurând un cod mai lizibil decât prin declararea separată de constante.

Putem folosi declaraţiile de tip typedef pentru a introduce un tip enumerare:

```
enum {D, L, Ma, Mc, J, V, S} zi;
/* tip anonim; declară doar variabila zi, tipul nu are nume, deci nu mai
putem declara altundeva variabile */

// sau:

typedef enum {D, L, Ma, Mc, J, V, S} Zi;
// tip enumerare cu numele Zi; se pot declara variabile

int nr_ore_lucru[7];          // vector cu număr de ore pe zi
for (zi = L; zi <= V; ++zi) nr_ore_lucru[zi] = 8;


typedef enum { inginer=1, profesor, avocat } profesie;
profesie profesia_mea=inginer; // definirea variabilei profesia_mea
```

IB.09.8. Exemple

1. Să se scrie un program care declară o strucură pentru un număr generic, folosind o uniune pentru valoarea numărului şi un câmp tip ce va spune ce fel de număr este (întreg – i, întreg lung – l, real – f, real dubla precizie – d).
Să se scrie o funcţie ce citeşte o variabilă de tipul structurii definite.
Să se scrie o funcţie ce afişează valoarea unei variabile de tipul structurii definite.

Rezolvare:

```
#include<stdio.h>

typedef struct numar {      // structura pentru un număr de orice tip
  char tipn;                // tip numar (caracter: i-int, l-long, f-float,
d-double
  union {
      int ival;
     long lval;
     float fval;
     double dval;
  } val;                      // valoare numar
} Numar;                      // definire tip de date Numar

// afisare număr
void write (Numar n) {
  switch (n.tipn) {                 // in functie de tip afiseaza valoarea
   case 'i': printf ("Intreg %d ", n.val.ival); break;
   case 'l': printf ("Intreg lung %ld ", n.val.lval); break;
   case 'f': printf ("Real %f ", n.val.fval); break;
   case 'd': printf ("Real dublu %.15lf ", n.val.dval);
  }
}

// citire număr
Numar read (char tip) {
  Numar n;
  n.tipn=tip;
  switch (tip) {
   case 'i': scanf ("%d", &n.val.ival); break;
   case 'l': scanf ("%ld", &n.val.lval); break;
   case 'f': scanf ("%f", &n.val.fval); break;
   case 'd': scanf ("%lf", &n.val.dval);
   }
  return n;
}

int main () {
  Numar a, b, c, d;
  a = read('i');
   b = read('l');
   c = read('f');
   d = read('d');
  write(a);
   write(b);
   write(c);
   write(d);
      return 0;
}
```

2. Pentru câmpul discriminant se poate defini un tip enumerare, împreună cu valorile constante (simbolice) pe care le poate avea. Să se refacă programul!

Rezolvare:

```
#include<stdio.h>

enum tnum {I, L, F, D};       // definire tip "tnum"

typedef struct{
```

```
  tnum tipn;                    // tip număr (un caracter)
  union {
      int ival;
     long lval;
     float fval;
     double dval;
  } val;                        // valoare numar
} Numar;                        // definire tip de date Numar

void write (Numar n) {
  switch (n.tipn) {
   case I: printf ("%d", n.val.ival); break;          // int
   case L: printf ("%ld", n.val.lval); break;         // long
   case F: printf ("%f", n.val.fval);break;           // float
   case D: printf ("%.15lf", n.val.dval);             // double
  }
}

Numar read (tnum tip) {
  Numar n;
  n.tipn=tip;
  switch (tip) {
       case I: scanf ("%d", &n.val.ival); break;
        ...
  }
  return n;
}

int main () {
      Numar a,b,c,d;
      a = read(I);
      write(a);
      ...
}
```