CURS 5
Tipuri structurate de date. Tipul enumerare.

5.1. Structuri

Frecvent, în practică, se dorește prelucrarea unor informații structurate, compuse din mai multe valori primitive.

O *structură* este o colecţie de valori eterogene stocată într-o zonă de memorie compactă. O structură este compusă din una sau mai multe elemente, numite câmpuri, în care fiecare câmp are propriul său nume şi tip. Memoria alocată unei structuri este o secvenţă continuă de locaţii, câte o locaţie pentru fiecare câmp. Câmpurile sunt numite şi membri ai structurii sau elemente ale structurii. Ordinea de memorare a câmpurilor corespunde cu ordinea de descriere a acestora în cadrul structurii.

Structurile permit organizarea datelor complexe, permiţând ca un grup de variabile legate să fie tratate ca o singură entitate. O structură are următorul *format general:*

```
struct nume_structura {
  tip1 camp1;
  tip2 camp2;
  ...
  tipN campN;
} variabila1, ..., variabilaN;
```

Observaţii:
1. O declaraţie de structură se termină cu ';', deoarece reprezintă definiția unor variabile.
2. O declaraţie de structură neurmată de o listă de variabile defineşte doar un tip structură, un şablon pe baza căruia se pot declara variabile de tipul structurii astfel definite.

Exemplu1:

```
struct punct {
 int x;
 int y;
} p;

int main ()
{
 struct punct alt_punct;
 printf ("Introduceti cele doua coordonate: ");
 scanf ("%d %d", &p.x, &p.y);
 p.x = p.x + 21;
 p.y = p.y + 10;
//între structuri se pot face atribuiri, se copiaza TOATE câmpurile
 alt_punct = p ;
 printf ("Coordonatele sunt: (%d %d) \n", alt_punct.x, alt_punct.y);
return 0;
}
```

Exemplul anterior declară o structura numită punct, alcătuită din două *câmpuri*: x si y, precum şi variabila p de acest tip. Orice variabilă de tip *struct punct* va putea memora coordonatele plane ale unui punct (exemplu variabila alt_punct). Membrii unei variabile structurate se acceseaza cu operatorul .(punct).

```
p.x = p.x + 21;
p.y = p.y + 10;
```

Exemplu2:

```
struct data_calendaristica {   //definim structura cu numele data_calendaristica
 int zi;
 int luna;
 int an;
 int secol;
 int mileniu;
} d;                           //variabila d va fi de tip structura data_calendaristica

struct data_calendaristica data_angajarii, data_nasterii;
                        //definim variabilele data_angajarii, data_nasterii de tip
                        // structura data_calendaristica
```

Numele complet al unui câmp se obţine din numele structurii urmat de caracterul "." (denumit operatorul punct) şi numele câmpului referit. Astfel pentru a referi câmpurile din structură (numite şi elemente sau membrii ai structurii) vom folosi d.zi, d.luna şi d.an. Câmpul selectat se comportă ca o variabilă şi i se pot aplica toate operaţiile care se pot aplica variabilelor de acel tip.

Un membru al unei structuri poate avea acelaşi nume cu o structură sau cu o variabilă oarecare, nemembru, deoarece nu se vor genera conflicte datorită faptului că numele de structuri se află într-un spaţiu de memorie separat de cel al numelor de variabile iar referirea unui membru al unei structuri respectiv referirea unei variabile se face diferit în cadrul unui program.

Exemplu tipic de structură este înregistrarea unui salariat. Aceasta poate conţine atribute precum marca salariatului, numele, prenumele, adresa, salarul brut, data naşterii, locul naşterii, data angajării etc. Se observă că structurile pot fi imbricate, adică membrii unei structuri pot fi la rândul lor structuri (este cazul datei angajării sau al datei naşterii, care sunt la rândul lor structuri):

```
struct angajat {
 int id;
 char nume[25];
 char prenume[25];
 struct data_calendaristica data_nasterii;
 struct data_calendaristica data_angajarii;
…
} ang;
```

Referirea în acest caz se face sub forma ang.data_nasterii.zi sau pentru un alt membru ang.data_angajarii.an.
Standardul ANSI C specifică faptul că structurile trebuie să permită imbricarea pe cel mult 15 niveluri.

Atunci când o structură este folosită ca argument al unei funcţii, întreaga structură este transmisă folosind metoda standard de apelare prin valoare. Aceasta înseamnă că orice modificare a conţinutului structurii în interiorul funcţiei căreia i se transmite ca parametru nu afectează structura utilizată ca argument. Dacă se doreşte adresa

unui membru individual al unei structuri, se foloseşte operatorul ‘&’ înaintea numelui structurii. Operatorul ‘&’ precede numele structurii, nu pe cel al membrului individual.

Exemplu:

```
functie(&ang.id, ang.nume);      //în cazul numelui nu este necesară utilizarea
                                 //operatorului ’&’ având în vedere că nume este deja o adresă
```

Se poate ca informaţia conţinută într-o structură să fie atribuită unei alte structuri de acelaşi tip folosind o singură instrucţiune de atribuire (deci nu trebuie atribuită valoarea fiecărui membru separat).

Exemplu:

```
struct angajat ang1, ang2;      //definim variabilele ang1 şi ang2 de tipul struct angajat
ang2=ang1;                      //atribuim valorile câmpurilor variabilei  ang1 câmpurilor
                                //corespondente din variabila ang2
```

Se pot defini masive de structuri.

```
struct angajat salariati[300];          //definim un masiv de 300 elemente, fiecare element
```
fiind de tip structură

Referirea unui element din acest masiv se face astfel:

```
int ln;
ln=salariati[5].data_nasterii.luna;          //variabila ln va conţine câmpul
                                             //data_nasterii.luna din al 6-lea element al structurii
```

5.2 Asignari de nume pentru tipuri de date

Tipurile de baza ale limbajului C, numite si tipuri predefinite se identifica printrun cuvant cheie (`int, char, float`). Tipurile structurate se definesc printr-o declaratie `struct nume {};`
Programatorul poate sa atribuie un nume unui tip, indiferent de faptul ca acesta este predefinit sau definit de utilizator. Aceasta se realizeaza prin intermediul constructiei typedef care are urmatorul format

```
typedef definitie_tip identificator;
```

Dupa ce s-a atribuit un nume unui tip, numele rspectiv poate fi utilizat pentru a declara date de acel tip, exact la fel cum se utilizeaza in declaratii cuvintele cheie `int, char, float, etc.`

Exemple:

1. Definitia: `typedef unsigned short USHORT;`

atribuie numele USHORT definitiei de tip unsigned short.
Deasemenea, daca dorim sa lucram cu siruri de lungime 40 sub numele str40 folosim definitia:

```
typedef char str40[41];
```

In contextul acestei definitii declaratia

```
str40 a,b;
```

defineste doua stringuri de lungime 41 de caractere. Astfel instructiunea
```
printf("\nsizeof(a) = %d\n",sizeof(a));
```
va tipari urmatoarele
```
sizeof(a) = 41
```

2. Declaratiile

```
typedef int INTREG;
typedef float REAL;
```

pot fi folosite ca si cuvintele cheie `int` si `float`. Astfel declaratia:
```
INTREG x,y,a[100];
```

este identica cu `int x,y,a[100];`

```
typedef struct data_calend { int zi;char luna[11];
     int an;
} Dc;
```
Prin aceasta declaratie se atribuie denumirea DC tipului structurat `data_calend.` In continuare putem declara date de tip DC:
```
DC data_angajarii, data_nasterii;
DC data_crt={20,"septembrie",1991};
```

6.3.Uniuni

O *uniune* este o locație de memorie care este partajată de două sau mai multe variabile, în general, de tipuri diferite. Sintaxa declaratiei unei uniuni este similara cu cea a structurii:

```
union nume_generic {
  tip nume_variabilă;
  tip nume_variabilă;
  …
} variabile_uniune;
```

unde identificatorii declarați ca membrii reprezintă nume cu care sunt referite obiectele de tipuri diferite care utilizează în comun zona de memorie.

Spațiul de memorie alocat corespunde tipului de dimensiune maximă. De exemplu, în secvența:

Tipurile uniune oferă posibilitatea unor conversii interesante, deoarece aceeasi zona de memorie poate conține informații organizate în moduri diferite, corespunzătoare tipurilor membrilor.

Exemplu:

```
union exemplu_uniune {
 int i;
 char ch;
};
union exemplu_uniune eu; //declarăm variabila eu de tipul uniune definit mai sus
```

Când este declarată o variabilă de tip uniune, compilatorul alocă automat memorie suficientă pentru a păstra cel mai mare membru al acesteia. Evidenţa dimensiunilor şi a aliniamentului o va ţine compilatorul.

În variabila eu atât întregul i cât şi caracterul ch împart aceeaşi locaţie de memorie, în care i ocupă 2 octeţi iar ch doar unul. Ne putem referi la datele stocate în variabila uniune definită atât ca la un caracter, cât şi ca la un întreg, din orice parte a programului. Sintactic, membri unei uniuni sunt accesibili prin construcţii de forma:

```
nume_uniune.membru sau pointer_la_uniune->membru.
```

Exemplu:

```
eu.i=10;
eu.ch='a';
```

Mărimea unei structuri sau a unei uniuni poate fi egală sau mai mare decât suma mărimilor membrilor săi. De aceea, când este necesară cunoaşterea mărimii unei structuri sau uniuni, se va folosi operatorul sizeof.

Uniunile pot apărea în structuri şi masive şi invers. Sintaxa pentru a apela un membru al unor astfel de structuri imbricate este aceeaşi (deci se foloseşte operatorul "." sau "-> " în cazul în care referirea se face folosind un pointer).

Observaţie: Este ilegal a declara o structură sau o uniune care face apel la ea însăşi, dar o structură sau uniune poate conţine un pointer la un apel spre ea însăşi.

6.4. Câmpuri de biţi

Limbajul C ofera posibilitatea de a structura datele la nivel de bit. Astfel, unor membri de structuri sau uniuni, li se pot aloca, dintr-un octet, biți individuali sau grupuri de biți. In felul acesta, se definesc câmpuri de biți care pot fi accesate fiecare, separate de restul octetului, pentru evaluare și/sau modificare.

Forma generală de definire a unei structuri cu membri de tip câmp de biți este:

```
struct nume_generic {
  tip nume1:lungime;
  tip nume2: lungime;
  …
  tip numen: lungime;
} lista_variabile;
```

Pentru campurile de biți există următoarele **restricții**:
- tipul poate fi *int*, *signed* sau *unsigned*;
- *lungime* este o constantă întreagă cu valori între 0 şi 15;
- nu se poate evalua adresa unui camp de biti;
- nu se pot organiza tablouri de campuri de biti;
- nu se poate ști cum sunt rulate câmpurile (de la dreapta la stânga sau invers, funcție de echipament).

Câmpurile de biti pot fi utile atunci cand informatia furnizata de anumite echipamente este transmisa prin octet sau atunci cand se doreste accesul la bitii unui octet sau atunci cand memoria este limitata si anumite informatiii pot fi stocate intr-un singur octet.

Câmpurile de biţi permit accesul la nivel de bit pentru elementele acestui tip de structură. Deoarece în C nu există tipul boolean, astfel se pot stoca mai multe variabile booleene (adevărat sau fals, 1 sau 0) utilizând un singur octet. De fapt, un câmp de biţi este un tip special de structură, care precizează cât de lung trebuie să fie fiecare câmp, lungime exprimată în biţi.

Exemplu:

```
struct angajat {
 struct date_pers datep;
 float salar;
 unsigned activ:1;
 unsigned orar:1;
} ang;
```

Se observă că este corect să amestecăm membri obişnuiţi ai unei structuri cu câmpuri de biţi. Astfel, vom defini o înregistrare care în loc să utilizeze 2 octeţi pentru a păstra informaţiile activ (angajat activ sau inactiv) şi orar (retribuţie lunară sau orară) va folosi doar 2 biţi pentru reţinerea celor 2 informaţii. Rezultă de aici o economie de memorie de 14 biţi (pentru acest caz) pentru fiecare element de structură de acest tip.

Nu este obligatorie numirea fiecărui element dintr-o structură de tip câmp de biţi. Dacă dorim utilizarea doar a ultimilor 2 biţi dintr-un octet, putem defini câmpul de biti astfel:

```
struct cb {
 unsigned:     6;
 unsigned bit7:     1;
 unsigned bit8:     1;
}
```

Nu se poate obţine adresa unui câmp de biţi, deoarece nu se ştie dacă aceste câmpuri vor fi rulate de la dreapta spre stânga sau invers, depinzând de compilator.

6.5. Enumerări

O *enumerare* este un set de constante care specifică toate valorile permise pe care le poate avea o variabilă de acel tip. Permite utilizatorului să folosească în program nume sugestive în locul unor valori numerice. De exemplu în locul numărului unei luni calendaristice se poate folosi denumirea ei sau în loc de 0 și 1 se poate folosi ADEVĂRAT sau FALS.
Formatul general de declarare a unei enumerări este:

```
enum nume_generic {lista enumerărilor} lista_variabile_enumerare;
```

– constantele pot avea specificate valori (și o valoare se poate repeta)

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

– implicit, șirul valorilor e crescător cu pasul 1, iar prima valoare e 0
– un nume de constantă nu poate fi folosit în mai multe enumerări
– tipurile enumerare sunt tipuri întregi, variabilele enumerare se pot folosi la fel ca variabilele întregi
– cod mai lizibil decât prin declararea separată de constante

```
enum {D, L, Ma, Mc, J, V, S} zi; // tip anonim; declară doar variabilă zi
                                 // tipul nu are nume , nu mai putem declara altundeva variabilele
int nr_ore_lucru[7]; //  număr de ore pe zi
for (zi = L; zi <= V; ++zi) nr_ore_lucru[zi] = 8;

enum curcubeu {rosu, orange, galben, verde, albastru, indigo, violet};
enum curcubeu ec;
printf("%d %d\n", rosu, verde);; //va afişa 0 3.
```

Asupra unei variabile enumerare se pot face atribuiri:

```
ec=albastru;
```

sau valoarea lor poate fi comparată cu una din valorile din enumerare:

```
if (ec==albastru) printf("Albastru!\n");
```

Pentru a specifica valoarea unuia sau mai multor simboluri putem folosi iniţializări. Simbolurilor care apar după iniţializare li se atribuie automat valori ce urmează după acea valoare.

```
enum curcubeu {rosu, orange, galben=100, verde, albastru, indigo, violet}
ec;
printf("%d %d\n", verde,violet); //va afişa 101 104.
```

Exemplu1:

```
typedef enum {verde, galben,rosu} culoare;
 culoare semafor;
 semafor=galben;
 ......
switch(semafor){
 case verde:libera_trecere();break;
 case galben: asteapta();break;
 case rosu: oprire_motor();break;
default: semafor_inactiv()
}
```

Exemplu2:

```
#include <stdio.h>
enum culori {ROSU, GALBEN, VERDE, ALBASTRU, VIOLET, NUMAR_CULORI};
typedef enum culori TCuloare;

int main ()
{
 TCuloare cer, padure;
 printf("In enum sunt %d culori\n",NUMAR_CULORI);
 cer=ALBASTRU;
 padure=VERDE;
 printf("cer=%d\n",(int)cer);
 printf("padure=%d\n",(int)padure);
return 0;
```

```
}
```
In mod natural, diferitele tipuri definite de utilizator pot fi combinate, astfel încât putem avea de exemplu tablouri de structuri sau invers, tablouri ale caror elemente sunt enumerări. Pentru a evita declarații de genul `struct node x,y,z` sau `enum boolean s,b`; limbajul C furnizează mecanismul de definire a numelui de tipuri cu ajutorul lui `typedef`. Acesta creează tipuri sinonime cu cele denumite și care pot fi utilizate în declarații de variabile, mărind claritatea programului. Este sugestivă definirea în programul anterior a tipului `TCuloare` și utilizarea lui în programul principal în locul tipului `enum culori`.