# 7 TIPURI DE DATE DEFINITE DE UTILIZATOR



## 7.1. TIPURI DEFINITE DE UTILIZATOR

Limbajele de programare de nivel înalt oferă utilizatorului facilități de a prelucra atât datele singulare (izolate), cât şi pe cele grupate. Un exemplu de grupare a datelor - de acelaşi tip - îl constituie tablourile. Datele predefinite şi tablourile (prezentate în capitolele anterioare) nu sunt însă suficiente. Informația prelucrată în programe este organizată, în general în ansambluri de date, de diferite tipuri. Pentru a putea descrie aceste ansambluri (structuri) de date, limbajele de programare de nivel înalt permit programatorului să-şi definească *propriile tipuri de date*.

Limbajul C oferă posibilități de definire a unor tipurilor de date, cu ajutorul:
- ***structurilor*** - permit gruparea unor obiecte (date) de tipuri diferite, referite printr-un nume comun;
- ***câmpurilor de biți*** - membri ai unei structuri pentru care se alocă un grup de biți, în interiorul unui cuvânt de memorie;
- ***uniunilor*** - permit utilizarea în comun a unei zone de memorie de către mai multe obiecte de diferite tipuri;
- declarațiilor ***typedef*** - asociază nume tipurilor noi de date;
- ***enumerărilor*** - sunt liste de identificatori cu valori constante, întregi.

## 7.2. STRUCTURI

Structurile grupează date de tipuri diferite, constituind definiții ale unor noi tipuri de date. Componentele unei structuri se numesc ***membrii (câmpurile)*** structurii. La declararea unei structuri se pot preciza tipurile, identificatorii elementelor componente şi numele structurii.
Forma generală de declarare a unei structuri:

```
struct identificator_tip_structura {
      lista_de_declaratii_membrii;
} lista_identificatori_variabile;
```

în care:

`struct` este un cuvânt cheie (obligatoriu)
`identificator_tip_structura` reprezintă numele noului tip (poate lipsi)
`lista_de_declaratii_membri` este o listă în care apar tipurile şi identificatorii membrilor structurii
`lista_identificatori_variabile` este o listă cu identificatorii variabilelor de tipul declarat.

Membrii unei structuri pot fi de orice tip, cu excepția tipului structură care se declară. Se admit însă, pointeri către tipul structură. `Identificator_tip_structura` poate lipsi din declarație, însă în acest caz, în `lista_identificatori_variabile` trebuie să fie prezent cel puțin un `identificator_varabila`. `Lista_identificatori_variabile` poate lipsi, însă, în acest caz, este obigatorie prezența unui `identificator_tip_structura`.
***Exemplu:*** Se defineşte noul tip de date numit `data`, cu membrii `zi`, `luna`, `an`. Identificatorii variabilelor de tipul `data` sunt `data_naşterii`, `data_angajării`.

```
struct data {
      int zi;
```

```
        char luna[11];
        int an;
} data_naşterii, data_angajării;
```

Declarația de mai sus poate apare sub forma:

```
struct data {
        int zi;
        char luna[11];
        int an;
};
struct data data_nasterii, data_angajarii;
/*Variabilele data_nasterii şi data_angajarii sunt date de tipul data */
```

Se poate omite numele noului tip de date:

```
struct {
        int zi;
        char luna[11];
        int an;
} data_naşterii, data_angajării;
```

Inițializarea variabilelor de tip nou, definit prin structură, se poate realiza prin enumerarea valorilor membrilor, în ordinea în care aceştia apar în declarația structurii. Referirea unui membru al structurii se realizează cu ajutorul unui operator de bază, numit ***operator de selecție***, simbolizat prin **.** .Operatorul are prioritate maximă. Membrul stâng al operatorului de selecție precizează numele variabilei de tipul introdus prin structură, iar membrul drept-numele membrului structurii, ca în exemplul următor:

***Exemplu:***

```
struct angajat{
        char nume[20], prenume[20];
        int nr_copii;
        double salariu;
        char loc_nastere[20];
};
struct angajat a1= {"Popescu", "Vlad", 2, 2900200, "Galati"};
a1.nr_copii = 3;
strcpy(a1.nume, "Popesco");
```

Variabilele de acelaşi tip pot apare ca operanzi ai operatorului de atribuire. În acest caz atribuirile se fac membru cu membru. În exemplul anterior am declarat şi inițializat variabila a1, de tip angajat. Declarăm şi variabila a2, de acelaşi tip. Dacă dorim ca membrii variabilei a2 să conțină aceleaşi valori ca membrii variabilei a1 (a1 si a2 de tip angajat), putem folosi operatorul de atribuire, ca în exemplul următor:

```
struct angajat a2;
a2=a1;
```

Aşa cum s-a observat din exemplul anterior, structurile pot avea ca membri tablouri (structura angajat are ca membrii tablourile de caractere loc_naştere[20], nume[20], prenume[20]). Deasemenea, variabilele de tip definit prin structură pot fi grupate în tablouri.

***Exemplu:***

```
struct persoana{
        char nume[20], prenume[20];
        int nr_copii;
        double salariu;
        char loc_nastere[20];
}angajati[100];
```

/* S-au declarat noul tip numit persoana şi variabila numită angajati, care este un vector (cu maxim 100 de elemente), ale cărui elemente sunt de tipul persoana */

//Inițializarea elementelor vectorului angajați[100]

```
for (int i=0; i<100; i++){
        cout<<"Intruduceti datele pentru angajatul "<<i+1<<'\n';
```

```
        cout<<"Numele :"; cin>>angajati[i].nume;
        cout<<"Prenumele :"; cin>>angajați[i].prenume;
        cout<<"Nr. copii:"; cin>> angajați[i].nr_copii;
        cout<<"Locul nașterii:"; cin>> angajați[i].loc_naștere;
}
```

Limbajul C permite definirea de structuri ale căror membri sunt tot structuri:

***Exemplu:***

```
struct data{
        int zi;
        char luna[11];
        int an;
};
struct persoana{
        char nume[20], prenume[20];
        int nr_copii;
        double salariu;
        char loc_naștere[20];
        struct data data_nașterii;
};
struct                                                          persoana
p1={"Popescu","Vasile",1,4000000,"Galati",{22,"Mai",1978}};
//Modificarea membrului data_nașterii pentru variabila p1 de tip persoana:
p1.data_nașteri.zi=23;
strcpy(p1.data_nașteri.luna, "Februarie");
p1.data_nasteri.an=1980;
```

Dacă se doreşte transmiterea ca parametri ai unor funcții a datelor de tip definit de utilizator prin structuri, acest lucru se realizează **numai** cu ajutorul pointerilor spre noul tipi.

De exemplu, este necesar ca variabila `p1`, de tip `persoana`, să fie prelucrată în funcția `f`, În acest caz, funcția va primi ca parametru un pointer spre tipul persoana. Funcția va avea prototipul:

```
void f(struct persoana *q);
```

Apelul funcției se realizează astfel: `f(&p1);`

În corpul funcției `f`, accesul la membrii varibilei q, de tip `persoana`, se realizează astfel:

```
(*q).nume;
(*q).prenume;
(*q).data_nașterii.an;
```
, etc.

Pentru a simplifica construcțiile anterioare, se foloseste ***operatorul de selecție indirectă*** (**->**):

```
q->nume;
q->prenume;
q->data_nașterii.an
```
, etc.

Structurile sunt utilizate în mod frecvent la definirea unor tipuri de date recursive (în implementarea listelor, arborilor, etc.). Un tip de date este direct recursiv dacă are cel puțin un membru care este de tip pointer spre el însuşi.

***Exemplu:***

```
struct nod{
        char nume[100];
        int an;
        struct nod *urmator;
};
```

***Exercițiu:*** Să se citească informațiile despre angajații unei întreprinderi, folosind o funcție de citire. Să se afişeze apoi informațiile despre angajați.

```
#include <stdio.h>
#include <conio.h>
struct persoana{
        char nume[20];int varsta;int salariu;
};
void cit_pers(struct persoana *ptr_pers)
{printf("Nume angajat:"); scanf("%s",ptr_pers->nume);
```

```
      printf("Varsta angajat:"); scanf("%d", &ptr_pers->varsta);
      printf("Salariu angajat:"); scanf("%d", &ptr_pers->salariu);
      }
      void main()
      {struct persoana *p;      //pointer catre date de tip persoana
      int nr_ang;  clrscr();
      printf("Nr. angajati:");scanf("%d", &nr_ang);
      p=new persoana[nr_ang]; //alocare dinamica a memoriei pentru cei nr_ang angajati
      for (int i=0; i<nr_ang; i++)
            cit_pers(&p[i]);
      printf("\n\n Datele despre angajati:\n\n");
      for (i=0; i<nr_ang; i++){
            printf("Angajatul %d\n NUME:  %s\n  VARSTA:  %d\n \ //continuare sir
              SALARIUL: %.d\n", i+1,p[i].nume,p[i].varsta, p[i].salariu);
            printf("\n\n  Apasa o tasta....\n"); getch();
            }
      }
```

Aşa cum se observă din exemplu, funcţia `cit_pers` primeşte ca parametru pointerul `ptr_pers`, către tipul `persoana`. Pentru a acesa membri structurii, în corpul funcţiei, se foloseşte operatorul de selecţie indirectă ($\rightarrow$). În funcţia `main`, se alocă memorie dinamic (cu ajutorul operatorului `new`). La afişare, în funcţia printf, şirul specificator de format se continuă pe rândul următor (folosirea caracterului \ pentru continuare).

## 7.3. CÂMPURI DE BIŢI

Limbajul C oferă posibilitatea de prelucrare a datelor la nivel de bit. De multe ori se utilizează date care pot avea doar 2 valori (0 sau 1), cum ar fi datele pentru controlul unor dispozitive periferice, sau datele de valori mici. Declarând aceste date de tip *int* sau *short int*, în memorie se rezervă 16 biţi. Alocarea unui număr atât de mare de locaţii de memorie nu este justificată, de aceea, limbajul C oferă posibilitatea declarării unor date pentru care să se aloce un număr specificat de biţi (alocare pe biţi).

***Definiţie:***

Un şir de biţi adiacenţi formeaza un ***câmp de biţi***.

Câmpurile de biţi se pot declara ca membri ai unei structuri, astfel:

```
struct identificator_tip_struct {
      tip_elem_1 identificator_elem_1:lungime1;
      tip_elem_2 identificator_elem_2:lungime2;
      . . .
      tip_elem_3 identificator_elem_3:lungime3;
} lista_identif_var_struct;
```

`Lungime1`, `lungime2`, etc. reprezintă lungimea fiecărui câmp de biţi, rezervat pentru memorarea membrilor. Câmpurile se alocă de la biţii de ordin inferior ai unui cuvânt (2 octeţi), către cei de ordin superior (figura 7.1).

***Exemplu:***

```
struct {
      int              a:    2;
      unsigned int     b:    1;
      int              c:    3;
} x, y;
```

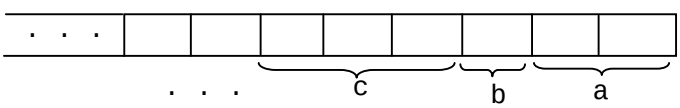


Figura 7.1. Câmpurile de biţi `a`, `b`, `c`

Câmpurile se referă ca orice membru al unei structuri, prin nume calificate:

***Exemplu:***

```
x.a = -1; x.b = 3; x.c = 4;
```

Utilizarea câmpurilor de biţi impune următoarele restricţii:

- Tipul membrilor poate fi `int` sau `unsigened int`.

- Lungime este o constantă întreagă din intervalul [0, 31];
- Un câmp de biți nu poate fi operandul unui operator de referențiere.
- Nu se pot organiza tablouri de câmpuri de biți.

Datorită restricțiilor pe care le impune folosirea câmpurilor de biți, cât şi datorită faptului că aplicațiile care folosesc astfel de structuri de date au o portabilitate extrem de redusă (organizarea memoriei depinzând de sistemul de calcul), se recomandă folosirea câmpurilor de biți cu precauție, doar în situațiile în care se face o economie substanțială de memorie.

## 7.4. DECLARAȚII DE TIP

Limbajul C permite atribuirea unui nume pentru un tip (predefinit sau utilizator) de date. Pentru aceasta se folosesc delcarațiile de tip. Forma generală a acestora este:

```
typedef tip nume_tip;
```

`Nume_tip` poate fi folosit la declararea datelor în mod similar cuvintelor cheie pentru tipurile predefinite.

***Exemplu:***

```
//1
      typedef int INTREG;
      INTREG x, y;
      INTREG z=4;
//2
      typedef struct{
            double parte_reală;
            double parte_imaginară;
      } COMPLEX;
      COMPLEX x, y;
```

## 7.5. UNIUNI

Aceeaşi zonă de memorie poate fi utilizată pentru păstrarea unor obiecte (date) de diferite tipuri, prin declararea uniunilor. Uniunile sunt similare cu structurile, singura diferență constând în modul de memorare. Declararea uniunilor:

```
union identificator_tip_uniune {
      lista de declaratii_membrii;
} lista_identificatori_variabile;
```

Spațiul de memorie alocat corespunde tipului membrului de dimensiune maximă. Tipul uniune foloseşte aceeaşi zonă de memorie, care va conține informații organizate în mai multe moduri, corespunzător tipurilor membrilor.

***Exemplu:***

```
union numeric{
      int i;
      float f;
      double d;
} num;
      num.i = 20;
      num.f = 5.80;
      cout<<sizeof(num)<<'\n';   //8
```

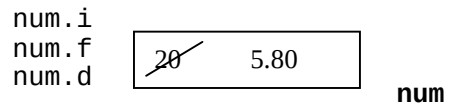


Figura 7.2. Modul de alocare a memoriei pentru variabila `num` (uniune) - 8 octeți

Pentru variabile num se rezervă 8 octeți de memorie, dimensiunea maximă a zonei de memorie alocate membrilor (pentru `int` s-ar fi rezervat 2 octeți, pentru `float` 4, iar pentru `double` 8). În exemplul anterior, în aceeaşi zonă de memorie se păstrează fie o valoare întreagă (`num.i=20`), fie o valoare reală, dublă precizie (`num.f=5.80`).

Dacă pentru definirea tipului numeric s-ar fi folosit o structură, modul de alocare a memoriei ar fi fost cel din figura 7.3.

```
struct numeric{
      int i;
      float f;
      double d;
} num;
      num.i = 20;
      num.f = 5.80;
      cout<<sizeof(num)<<'\n';   //14
```

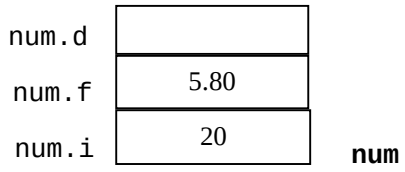


Figura 7.3. Modul de alocare a memoriei pentru variabila num (structură) - 14 octeți

## 7.6. ENUMERĂRI

Tipul enumerare asociază fiecărui identificator o consatantă întreagă. Sintaxa declarației:

```
enum identificator_tip_enumerare {
      identif_elem1 = const1, . . .
} lista_identif_variabile;
```

Din declarație pot lipsi fie `identificator_tip_enumerare`, fie `lista_identif_variabile`. Pentru fiecare element al enumerării, constanta poate fi asociată în mod explicit (ca în declarația anterioară), fie implicit. În modul implicit nu se specifică nici o constantă, iar valoarea implicită este 0 pentru primul element, iar pentru restul elementelor, valoarea precedentă incrementată cu 1. Enumerările se folosesc în situațiile în care variabilele pot avea un număr mic de valori întregi, asociind un nume sugestiv pentru fiecare valoare.

***Exemplu:***

```
//1
      enum boolean {FALSE, TRUE};     //definirea tipului boolean cu elementele FALSE si TRUE
      //declaratie echivalenta cu  enum boolean {FALSE=0, TRUE=1};
      cout<<"FALSE este "<<FALSE<<'\n';     //FALSE este 0
//2
      typedef enum temperatura {mica=-10, medie=10, mare=80};
      //tipul enumerare temperatura, cu elementele mica (de valoare -10), medie (valoare 10), mare
      (valoare 80)
      temperatura t1, t2;       //declararea variabilelor t1, t2 de tip enumerare temperatura
      t1=medie;
      cout<<"t1="<<t1<<'\n';   //t1=10
```

***Exercițiu:*** Să se citească (cu ajutorul unei funcții de citire) următoarele informații despre elevii participanți la un concurs de admitere: nume, numărul de înscriere şi cele trei note obținute. Să se afişeze, printr-o funcție, informațiile citite. Să se afişeze o listă cu elevii participanți la concurs, ordonați alfabetic, notele şi media obținută (funcție de ordonare, funcție de calculare a mediei). Să se afişeze lista elevilor înscrişi la concurs, în ordinea descrescătoare a mediilor.

Sunt prezentate câteva modalități de implementare. În aceste variante apar doar funcția `cit_elev` (de citire) şi `main`. S-a definit tipul `elev`. Se lucrează cu vectori de tip elev. În funcția `cit_elev` se validează fiecare notă. Se va observa modul de acces la membri structurii în funcția `cit_elev`. Dezavantajul principal al acestui mod de implementare îl constituie risipa de memorie, deoarece în funcția `main` se rezervă o zonă de memorie continuă, pentru 100 de elemente de tip elev (100*sizeof(elev)).

```
#include <iostream.h>
#include <conio.h>
typedef struct elev{
      char nume[20];int nr_matr;int note[3];
};     //definirea tipului elev
void cit_elevi(elev a[], int n)
```

```
{for (int i=0; i<n; i++){
      cout<<"Nume elev:"; cin>>a[i].nume; //citirea numelui unui elev
      cout<<"Nr. insriere:";  cin>>a[i].nr_matr;
for (int j=0; j<3; j++){      // citirea notelor obtinute
            do{
                  cout<<"Nota :"<<j+1<<" =";cin>>a[i].note[j];
                  if   (a[i].note[j]<0 || a[i].note[j]>10)  //validarea notei
                        cout<<"Nota incorecta!....Repeta!\n";
            }while (a[i].note[j]<0 || a[i].note[j]>10);
      }
  }
}
void main()
{ int nr_elevi;  clrscr();
cout<<"Nr. elevi:";cin>>nr_elevi;
elev p[100];      //declararea tabloului p, de tip elev
cit_elevi(p, nr_elevi); //apel functie
}
```

În varianta următoare, se lucrează cu pointeri către tipul `elev`, iar memoria este alocată dinamic.

```
typedef struct elev{
      char nume[20];int nr_matr;int note[3];
};    //definirea tipului elev
void cit_elevi(elev *a, int n)
{
for (int i=0; i<n; i++){
      cout<<"Nume elev:"; cin>>(a+i)->nume; //sau cin>>(*(a+i)).nume;
      cout<<"Nr. insriere:";  cin>>(a+i)->nr_matr;
      for (int j=0; j<3; j++){
            do{
            cout<<"Nota :"<<j+1<<" =";
            cin>>(a+i)->note[j];
            if   ((a+i)->note[j]<0 || (a+i)->note[j]>10)
                  cout<<"Nota incorecta!....Repeta!\n";
            }while ((a+i)->note[j]<0 || (a+i)->note[j]>10);
      }
  }
}
void main()
{ int nr_elevi;  clrscr();
cout<<"Nr. elevi:";cin>>nr_elevi;
elev *p;     //declararea pointerului p, către tipul elev
p=new elev[nr_elevi];
//alocarea dinamică a memoriei, pentru un tablou cu nr_elevi elemente
cit_elevi(p, nr_elevi); //apel functie
}
```

Implementarea tuturor funcțiilor:

```
#include <stdio.h>
#include <string.h>
#define DIM_PAG   24     //dimensiunea paginii de afisare
#define FALSE 0
#define TRUE 1
void ord_medii(elev *a, int n)
{
int gata =FALSE;int i;double med1, med2;elev aux;
while (!gata){
      gata=TRUE;
      for (i=0; i<=n-2; i++){
          med1=0;med2=0;
```

```
            for (int j=0; j<3; j++){
              med1+=(a+i)->note[j]; med2+=(a+i+1)->note[j];
              //calculul mediilor pentru elementele vecine
            }
            med1/=3; med2/=3;
              if (med1<med2){
                    aux=*(a+i); *(a+i)=*(a+i+1);*(a+i+1)=aux;
                    gata=FALSE; }
      }
 }
}
void ord_alf(elev *a, int n)
{
int gata =FALSE;int i;double med1, med2;elev aux;
while (!gata){
      gata=TRUE;
      for (i=0; i<=n-2; i++){
            if (strcmp( (a+i)->nume,(a+i+1)->nume) >0){
                  aux=*(a+i); *(a+i)=*(a+i+1);*(a+i+1)=aux;
                  gata=FALSE;}
      }
 }
}
void cit_elevi(elev *a, int n);
// functie implementata anterior
void antet_afis(const char *s)
{printf("%s\n", s);
}
void afis_elev(elev *a, int n, char c)
{clrscr();
if (c=='A')
      antet_afis("      LISTA INSCRISILOR \n");
if (c=='O')
      antet_afis("      LISTA ALFABETICA  \n");
if (c=='R')
      antet_afis("      LISTA MEDII       \n");
printf("Nr.crt.|Nr. Matricol|              NUME        |Nota1|Nota2|Nota3|
MEDIA\  |\n");
printf("-------------------------------------------------------------------\
\n");
int lin=3;
for (int i=0; i<n; i++){
      printf("%7d|%12d|%-20s|",i+1,(a+i)->nr_matr,(a+i)->nume);
      double med=0;
      for (int j=0; j<3; j++){
            printf("%-5d|", (a+i)->note[j]);
            med+=(a+i)->note[j];
      }
      med/=3;printf("%-9.2f|\n", med);lin++;
      if (lin==(DIM_PAG-1)){
            printf("  Apasa o tasta...."); getch();
            clrscr();
            if (c=='A') antet_afis("      LISTA INSCRISILOR \n");
            if (c=='O') antet_afis("      LISTA ALFABETICA  \n");
            if (c=='R') antet_afis("      LISTA MEDII       \n");
            printf("Nr.crt.|          NUME      |Nota1|Nota2|Nota3| MEDIA\
|\n");
            printf("-----------------------------------------------------\
\n");
            int lin=3;
        }
  }
```

```
        printf("  Apasa o tasta...."); getch();
}
void main()
{ int nr_elevi;  clrscr();
cout<<"Nr. elevi:";cin>>nr_elevi;
elev *p; p=new elev[nr_elevi];
cit_elevi(p, nr_elevi);
afis_elev(p, nr_elevi, 'A');//afisarea inscrisilor
ord_medii(p, nr_elevi);
afis_elev(p,  nr_elevi,  'R');//afisarea  in  ordinea  descrescatoare  a
mediilor
ord_alf(p, nr_elevi);  //ordonare alfabetica
afis_elev(p,  nr_elevi,  'O');//afisarea  in  ordinea  descrescatoare  a
mediilor
}
```

S-au implementet următoarele funcții:
`cit_elevi` - citeşte informațiile despre elevii înscrişi.
`afis_elevi` - afişează informațiile despre elevi. Această funcție este folosită pentru cele trei afişări (lista înscrişilor, lista alfabetică şi clasamentul în ordinea descrescătoare a mediilor). Afişarea se realizează cu ajutorul funcției `printf`, care permite formatarea datelor afişate. Afişarea se realizează ecran cu ecran (se foloseşte variabila `lin` care contorizează numărul de linii afişate), cu pauză după fiecare ecran. La începutul fiecărei pagini se afişează titlul listei - corespunzător caracterului transmis ca parametru funcției - şi capul de tabel. Deasemenea, pentru fiecare elev înscris se calculează media obținută (variabila `med`).
`ord_medii` - ordonează vectorul de elevi (transmis ca parametru, pointer la tipul elev), descrescător, după medii. Se aplică metoda BubbleSort, comparându-se mediile elementelor vecine (`med1` reprezintă media elementului de indice `i`, iar `med2` - a celui de indice `i+1`) ale vectorului.
`ord_alf` - ordonează vectorul de elevi (transmis ca parametru, pointer la tipul elev), crescător, după informația conținută de membrul `nume`. Pentru compararea numelor se foloseşte funcția `strcmp`.
Deoarece este foarte probabil ca vectorul înscrişilor să aibă multe elemente, pentru ordonări, ar fi fost mai eficientă metoda `QuickSort`; s-a folosit BubbleSort pentru a nu complica prea mult problema.

## ÎNTREBĂRI ŞI EXERCIȚII

### Chestiuni teoretice

1. Variabilele tablou şi variabilele de tip definit de utilizator sunt exemple de variabile compuse (reprezintă date structurate). Care este, totuşi, deosebirea dintre ele?
2. Ce posibilități de definire a unor noi tipuri de date vă oferă limbajul C/C++?
3. În ce constă diferența dintre structuri şi uniuni?
4. Cum se numesc componentele unei structuri?
5. Ce restricții impune folosirea câmpurilor de biți?
6. Există vreo restricție referitoare la tipul membrilor unei structuri? Dacă da, care este aceasta?

### Chestiuni practice

1. Să se implementeze programele cu exemplele prezentate.
2. Să se scrie programele pentru exercițiile rezolvate care au fost prezentate.
3. Realizați următoarele modificări la exercițiul prezentat la sfârşitul capitolului:

a) Completați cu o funcție de calcul şi afişare a mediei notelor tuturor candidaților pentru fiecare probă (media tuturor elevilor la proba1, media la proba2, etc).
b) Modificați lista alfabetică, astfel încât la elevii cu medie peste 5, să apară (alături de medie) mesajul "Promovat", iar la ceilalți, mesajul "Nepromovat".

c) Considerând că rezultatelor obținute sunt utilizate la un concurs de admitere, la care există N locuri (N introdus de la tastatură), şi de faptul că pentru a fi admis media trebuie să fie cel puțin 5, să se afişeze lista admişilor şi lista respinşilor, în ordinea descrescătoare a mediilor, în limita locurilor disponibile.

2. Să se scrie un program care să permită memorarea datelor privitoare la angajații unei firme mici: nume angajat, adresă, număr copii, sex, data naşterii, data angajării, calificare, salariul brut. Se vor implementa următoarele funcții:

a) Citirea informațiilor despre cei N angajați (N introdus de la tastatură);
b) Căutarea - după nume - a unui angajat şi afişarea informațiilor despre acesta;
c) Modificarea informațiilor despre un anumit angajat;
d) Lista alfabetică a angajaților, în care vor apare: nume, adresă, data angajării, calificare, salariu;
e) Lista angajaților în ordone descrescătoare a vechimii;
f) Lista angajatilor cu un anumit numar de copii, C, introdus de la tastatură;
g) Lista angajaților cu vârsta mai mare decât V (V introdus de la tastatură);
h) Salariul minim, salariul mediu şi cel maxim din firmă;
i) Lista de salarii, în care vor apare: numele, calificarea, salariul brut şi salariul net. La sfârşitul listei vor apare totalurile pentru salariile brute, impozite, salarii nete. Pentru calculul salariului net se aplică următoarele reguli de impozitare:
   i.1) *I=15%* pentru salariul brut (SB)<600000
   i.2) *I=50000+20%* pentru 600000<=SB<1500000 (20% din ceea ce depăşeşte 600000)
   i.3) *I=100000+30%* pentru 1500000<=SB<3000000
   i.4) *I=250000+40%* pentru 3000000<=SB<15000000
   i.5) *I=45%* pentru SB>=1500000