

# Programarea Calculatoarelor

## Laborator 4

### 1. Structuri de date

În C se pot defini tipuri structură (**struct**), enumerare (**enum**) și uniune (**union**).

- **Struct**

```
1. [typedef] struct [nume_tip_nou] {
2.     tip_de_date  camp_1;
3.     tip_de_date  camp_2;
4.     .....
5.     tip_de_date  camp_n;
6. } [lista_identificatori];
```

*Exemple:*

```
1. struct angajat
2. {
3.     char cnp[14], nume[50], data_ang[11], post_ocupat[30];
4.     float salariu;
5.     int zile_concediu, nr_copii;
6. };
7. int main()
8. {
9.     struct angajat a1;
10.    scanf("%d",&a1.zile_concediu);
11.    .....
```

```
• typedef struct
• {
•     char cnp[14], nume[50], data_ang[11], post_ocupat[30];
•     float salariu;
•     int zile_concediu, nr_copii;
• } angajat;
• int main()
• { angajat a1;
•   angajat a2={"1841211305600","Radu M. ","11.12.2013","Analist",5300.45,20,2};
•   printf("%d",a1.zile_concediu);
•   .....
```

```
1) typedef struct
2) {
3)     char nume[50];
4)     int semigrupa;
5) } student;
6) int main()
7) {
8) student s[3]={{"Ion A.",1331},{"Vlad R.",1341},{"Dinu E.",1432}};
9) printf("%d",s[2].semigrupa);
10) .....
```

#### Observații

1. Câmpurile unei structuri pot fi de orice tip de date, dar nu de tipul aceleiasi structuri (recursiv);
2. Structuri diferite pot conține câmpuri identice (fără conflict);

3. Structurile pot fi transmise/returnate în funcții;
4. Nu se pot compara folosind operatori logici.

## 2. Uniuni (*union*)

O uniune este un tip special de date care permite stocarea diferitelor tipuri de date în aceeași locație de memorie. Poate avea mai mulți membri, însă un singur membru poate conține o valoare la un moment dat. Uniunile ne permit să utilizăm în mod eficient aceeași locație de memorie în mai multe scopuri. Atenție cum sunt folosite.

Declararea unei uniuni este similară cu cea a unei structuri:

```
[typedef] union [nume_generic] {  
    tip nume_1;  
    tip nume_2;  
    .....  
    tip nume_n;  
} [lista_variabale];
```

### Observații:

1. Putem avea uniuni anonime (fără **nume\_generic**) .
2. Pentru a declara variabile de tip **nume\_generic** folosim construcția:  
`union nume_generic variabila_union;`

sau adăugăm numele variabilelor separate prin virgulă înainte de punctul și virgulă finală (în **lista\_variabale**) .

3. Rulați următorul exemplu:

```
union exemplu  
{  
    int nr;  
    long long v;  
} z;  
  
scanf("%d",&z.nr);  
printf("%d      %ld",z.nr,z.v);
```

4. Dimensiunea unei uniuni va fi suficient de mare ca cel mai mare membru al ei.

*Exemplu:*

```
union alfa {  
    char ch[3];  
    int y;  
} beta;  
printf("Memoria ocupata de beta este de %d octeti", sizeof(beta));
```

5.

```
int main() {  
    union alfa gamma;  
    gamma.y = 3;  
    printf("gamma.y: %d", gamma.y); // gamma.y: 3  
    strcpy(gamma.ch, "Da");  
    printf("gamma.ch: %s", gamma.ch); //gamma.ch : Da  
    return 0;  
}
```

6. Rulați următoarele instrucțiuni

```
int main() {
```

```

union alfa gamma;
gamma.y = 3;
strcpy(gamma.ch, "Da");

printf("gamma.y: %d", gamma.y);      //ce observati?
printf("gamma.ch: %s", gamma.ch);
return 0;
}

```

## 2. Enumerari (*enum*)

O enumerare este o multime de constante de tip intreg care reprezinta toate valorile permise pe care le poate avea o variabila de acel tip.

Declarare:

```

enum [nume_generic] {
    constanta_1,
    constanta_2,
    ....
    constanta_n
} [lista_variabile];

```

### Observații:

12. Implicit, sirul valorilor constantelor e crescator cu pasul 1, iar prima valoare este 0.

```

enum saptamana {
    Luni,
    Marti,
    Miercuri,
    Joi,
    Vineri,
    Sambata,
    Duminica
} zi;

int main()
{
    for(zi = Luni; zi <= Duminica; zi++) {
        printf("%d ", zi);
    }
    return 0;
}

```

### Testați!

13. Putem atribui si alte valori identificatorilor din sirul constantelor decat cele implicite, caz in care identificatorul urmator va avea valoarea corespunzatoare celui precedent + 1

```

enum culori {
    alb=-5,
    rosu,    // -4
    verde,   // -3
    albastru = 8, //8
    negru    //9
} culoare;

```

### Testați!

```

for(culoare = alb; culoare <= negru; culoare++) {
    printf("%d ", culoare);
}

```

14. Un identificator dintr-o enumerare este unic (nu poate aparea intr-o alta enumerare).
15. Memoria ocupata: cat pentru **int**.

16. De ce **enum**, in locul lui **#define** sau **const**?

*Exemplu:* Inserati **dir\_HR** pe pozitia a 3-a in lista:

```
#define director_gen 1
#define dir_AST 2
#define dir_SMN 3
#define dir_CRM 4
#define dir_TLN 5
```

### 3. Campuri de biti

Un camp de biti este un membru special al unei structuri, caruia i se specifica si numarul efectiv de biti.

Declarare:

```
struct [nume_generic] {
    tip nume_1 : lungime;
    tip nume_2 : lungime;
    ....
    tip_nume_n : lungime;
}[lista_variabile];
```

#### Observatii:

1. Daca un camp de biti **nu** este specificat ca **unsigned**, atunci bitul cel mai semnificativ este bitul de semn.

*Exemplu:* Un câmp definit pe 3 biți cu modificatorul **unsigned** va reține valori între 0 și 7.

Dacă nu apare modificatorul **unsigned**, atunci câmpul este cu semn și va reține valori între -4 și 3. **De ce?**

2. Într-o structură pot alterna câmpurile definite pe biți cu cele definite clasic.

*Exemplu:*

```
struct cofetarie {
    unsigned tip: 6;
    float pret;
    unsigned short nr_Euri: 3;
} c1;
```

Nu se poate accesa adresa unui camp al structurii pentru care avem specificat numarul de biți:

```
printf("%x",&c1.pret); // e ok;
printf("%x",&c1.tip); // da eroare;
```

3. *Exemplu:*

```
typedef struct {
    unsigned short camera : 4; // pana la 15 camere
    unsigned short ocupat: 1; // ocupat 1, liber 0
    unsigned short platit : 1; // platit 1, restanta 0
    unsigned short perioada_inchiriere : 2; // perioada //in luni
} camin;
```

```
camin grozavesti,kogalniceanu[2];
printf("%d %d %d",sizeof(camin),sizeof(grozavesti),sizeof(kogalniceanu));
```

#### Observatii:

- **Verificați memoria ocupată!**
  - Fara a utiliza campuri pe biti, ar fi fost necesari 8 octeti.
4. Accesarea membrilor este aceeași ca in cazul structurilor:
- ```
kogalniceanu[0].camera = 10;
```

5. Daca incercam sa atribuim unui camp mai o valoare ce ocupa mai mult decat numarul specificat de biti, acest lucru nu va fi permis:

```
kogalniceanu[1].camera = 20;
printf("camera %d", kogalniceanu[1].camera);
```

6. Fie următoarele declarații:

a) struct vietate

```
{
    unsigned short tip: 2; // 0 - pasare; 1 - peste; 2-patruped; 3-sarpe
    float greutate; // greutatea in kg */
    char nume[40];      //denumirea latinească
    unsigned short viteza: 6; // intre 1 si 63 km/ora
} x;
```

b) struct vietate

```
{
    unsigned short tip: 2; // 0 - pasare; 1 - peste; 2-patruped; 3-sarpe
    unsigned short viteza: 6; // intre 1 si 63 km/ora
    float greutate; // greutatea in kg */
    char nume[40];      //denumirea latinească
} x;
```

### Verificați memoria ocupată!

7. Fie declarația:

```
struct exemplu {
    int : 2;                // primii doi biți nu interesează
    int eroare: 1;          // un bit, semnalizează o eroare
    int status: 3;          // un câmp pe 3 biți
    int : 0;                // forțează alinierea la octetul următor
    int secventa: 4;        // număr de secvență pe 4 biți
} z;
```

## Probleme

1. a) Să se construiască o structură ce conține următoarele date despre candidații la admitere: nr\_legitimatie, nume, nota\_mate, nota\_info, nota\_bac, medie, admis (Y/N), buget (Y/N).

b) Sa se defineasca o instrucțiune ce calculează media de admitere dupa regula: 80% media la examen, 20% media de bac. Promovabilitatea se calculeaza utilizand pragul de 5.

c) Sa se scrie o funcție care citește datele unui candidat, in afara de medie, admis si buget, si le adauga unui vector al tuturor candidatilor, pastrand ordinea alfabetica. Media si promovabilitatea vor fi calculate folosind definitiile de la punctele b). Numarul de candidati este citit de la tastatura.

*Observatie: se va folosi functia **strcmp**(s1,s2) ce returneaza un numar:*

*o negativ, daca s1 este mai mic decat s2 dpdv al continutului;*

*o zero, daca s1 este identic cu s2;*

*o pozitiv, daca s1 este mai mare decat s2 dpdv al continutului.*

e). Să se scrie o funcție care completează campul „buget” cu Y sau N după regula: primii 75% (rotunjit în jos) dintre candidații admiși, în ordinea mediilor, sunt la buget (Y), restul la taxa (N) sau nu au promovat examenul de admitere (lasați campul gol).

f) Să se scrie o funcție care afișează datele candidaților în funcție de opțiunea aleasă: toți candidații (alfabetic), cei admiși la buget, cei admiși la taxa, cei respinși (ordonati descrescător după medie). (menu cu switch)

2. Să se construiască o structură de date potrivită pentru a memora o matrice rară (matrice de dimensiune  $n \times m$ ,  $1 \leq n, m \leq 50000$ , numărul elementelor nenule  $1 \leq k \leq 100$ ). Să se scrie câte o funcție pentru adunarea și înmulțirea a două matrici rare. Elementele matricei se vor citi ca triplete (l,c,x), unde l=linie, c=coloană, x=elementul nenul, în ordine crescătoare după linie și apoi după coloană. Afișarea se va face ca matrice (pe linii și coloane).

### Observații:

- Nu se vor folosi alte funcții de lucru cu siruri de caractere în afara de strcmp și strcpy;
- Nu se vor folosi pointeri;
- Toate afișările trebuie să conțină mesaje corespunzătoare.

3. Să se implementeze o funcție care folosește o uniune pentru a inversa cei doi octeți ai unui întreg (reprezentat pe 2 octeți) citit de la tastatură. Programul principal va apela funcția pentru a codifica și decodifica un întreg dat.

Exemplu:  $n = 20 \rightarrow 20$  codificat este 5120

5120 decodificat este 20

4. Folosind o singură structură, numită locuință, memorați următoarele date:

- adresa (cel mult 100 de caractere);
- suprafața;
- tip locuință (sir de cel mult 30 caractere): „garsoniera”, „casa” sau „apartament”;
- nr camere;
- în funcție de tipul de locuință, să reținem:
  - pentru garsoniera: balcon/nu (1/0);
  - apartament: de comandat/necomandat (D/N);
  - casa: sir de caractere - una din variantele: „pe sol”, „parter+mansarda”, „nr etaje”;

Cerinte:

1. Citiți datele a **n** locuințe;
2. Afișați adresa garsonierei ce are balcon și totodată cea mai mare suprafață.

5. Definiți o structură pentru memorarea următoarelor informații despre angajații unei firme:

- vârstă: sub 65 de ani;
- nume: maxim 30 de caractere;
- normă întreaga/part-time;
- CNP.

Cerinte:

1. Definiți structura în așa fel încât să ocupe spațiul minim de memorie posibil. Afișați spațiul de memorie ocupat, folosind operatorul sizeof.
2. Folosind structura definită, citiți de la tastatură informații despre un angajat, apoi afișați numai bărbații din firmă, mai tineri de 31 de ani (verificați vârsta folosind operatorii pe biți).

6. Definiți o structură de date ce ocupa spațiu minim, potrivita pentru a memora informatia daca un angajat a adus sau nu in dosarul firmei urmatoarele acte:

- Copie buletin;
- Copie certificat casatorie;
- Copie diploma licenta;
- Copie diploma master;
- Copie diploma doctor;
- Fisa de lichidare de la locul de munca anterior;
- Certificate de nastere copii – pentru deducere impozit.