Programare procedurală

– Laborator 4 –

Recapitulare Structuri (din laboratorul 3):

```
[typedef] struct [nume_tip_nou] {
        tip_de_date camp_1;
        tip_de_date camp_2;
        ……….
        tip_de_date camp_n;
        } [lista_identificatori];
```

I. Uniuni *(union)*

O uniune este un tip de date care permite stocarea unor valori, posibil având tipuri de date diferite, la aceeasi locație de memorie de memorie. Poate avea mai mulți (multe) membri (câmpuri), însă un singur membru (câmp) conține la un moment dat o valoare care nu este afectată de valorile celorlalți (celorlalte) membri (câmpuri). Modificarea valorii unui membru (câmp) afectează valorile celorlalți (celorlalte) membri (câmpuri). Uniunile ne permit să utilizăm în mod eficient aceeași locație de memorie în mai multe scopuri. În momentul utilizării unei variabile de tip uniune, trebuie verificat cu atenție faptul că membrul (câmpul) respectiv are atribuită o valoare corespunzătoare.

Exemplu:

```
typedef union
{
    int x;
    double y;
}Test;

int main(){
Test t;
t.y = 3.14;
t.x++;
printf("\nt.x = %d\nt.y = %f\n\n", t.x, t.y);
return 0;
}
```

Declararea unei uniuni este similară cu cea a unei structuri:

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

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 adaugăm numele variabilelor separate prin virgula înainte de punctul și virgula finală (în **lista_variabile**).

3. Rulați următorul exemplu:

```
union exemplu
{
    int nr;
    long long v;
} z;

scanf("%d",&z.nr);
printf("%d %lld",z.nr,z.v);
```

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

Exemplu:

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

5.

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

II. Enumerări *(enum)*

O enumerare este o mulțime de constante simbolice de tip întreg.

Declarare:

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

Observații:

1. Implicit, sirul valorilor constantelor e crescător 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!

2. O variabilă de tip enumerare nu are domeniul de valori restricționat la setul de valori declarat în enumerare, ci se comportă ca o variabilă de tip întreg obișnuită!

   Exemplu:

```
zi = Vineri;
printf("\nZiua = %d\n", zi);
zi = zi + 10;
printf("\nZiua = %d\n", zi);
```

   Se va afișa 4 și 14, fără a fi semnalizată nicio eroare!!!

3. Un identificator dintr-o enumerare este unic (nu poate aparea intr-o alta enumerare).

4. Memoria ocupată: cât pentru **int.**

5. Putem atribui și alte valori identificatorilor din șirul constantelor decât cele implicite, caz în care identificatorul următor va avea valoarea corespunzatoare valorii precedente + 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);
     }
```

6. 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
```

III. Câmpuri de biți

Un câmp de biți este un membru special al unei structuri, căruia i se specifică și numărul efectiv de biți.

Declarare:

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

Observații:

1. Dacă un câmp de biți **nu** este specificat ca **unsigned,** atunci bitul cel mai semnificativ este bitul de semn.

Exemplu: Un câmp definit pe 3 biţi cu modificatorul `unsigned` va reține valori între 0 și 7. Dacă nu apare modificatorul `unsigned`, atunci câmpul este cu semn şi va reține valori între -4 și 3. **De ce?**

2. Într-o structură pot alterna câmpurile definite pe biţi cu cele definite clasic.
   Exemplu:

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

Nu se poate accesa adresa unui câmp al structurii pentru care avem specificat numărul 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; // in luni
     } camin;

camin grozavesti,kogalniceanu[2];
printf("%ld  %ld   %ld \n",
sizeof(camin),sizeof(grozavesti),sizeof(kogalniceanu));
```

Observații:

- **Verificați memoria ocupată!**
- Fără a utiliza câmpuri pe biți, ar fi fost necesari 8 octeți.

4. Accesarea membrilor este aceeasi ca în cazul structurilor:

```
kogalniceanu[0].camera = 10;
```

5. Dacă încercăm să atribuim unui câmp o valoare ce ocupă mai mult decât numărul specificat de biți, acest lucru nu va fi semnalizat ca o eroare, efectul depinzând de implementarea compilatorului:

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

În general, efectul va fi același ca și în cazul variabilelor obișnuite, adică va calcula modulo valoarea maximă a tipului de date respective (în exemplul de mai sus se afișează camera ca fiind 4, deoarece 20 % 16 = 4).

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 latineasca
     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 latineasca
} x;
```

Verificați memoria ocupată!

7. Fie declarația:

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

Verificați memoria ocupată!

Probleme

1. Să se implementeze o functie care foloseşte o uniune pentru a inversa cei doi octeti ai unui întreg (reprezentat pe 2 octeți) citit de la tastatura. Programul principal va apela funcția pentru a codifica și decodifica un întreg dat.
   Exemplu: n = 20 ➔ 20 codificat este 5120;
   5120 decodificat este 20.

2. Folosind o singură structură, numită locuinta, memorați următoarele date:
   - adresa (cel mult 100 de caractere);
   - suprafața;
   - tip locuință (sir de cel mult 30 caractere): "garsoniera", "casa" sau "apartament";
   - număr camere;
   - în funcție de tipul de locuință, să reținem:
     - pentru garsonieră: balcon/nu (1/0);
     - apartament: decomandat/nedecomandat (D/N);
     - casă: șir de caractere - una din variantele: "pe sol", "parter+mansarda", "nr etaje";

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

3. Definiţi o structură pentru memorarea următoarelor informaţii despre angajatii unei firme:
   - vârsta: sub 65 de ani;
   - nume: maxim 30 de caractere;
   - normă întreaga/part-time;
   - CNP.

   Cerinte:
   a. Definiţi structura în aşa fel încât să ocupe spaţiul minim de memorie posibil. Afişaţi spaţiul de memorie ocupat, folosind operatorul sizeof.
   b. Folosind structura definită, citiţi de la tastatură informaţii despre un angajat, apoi afișați numai bărbații din firmă, mai tineri de 31 de ani (verificați vârsta folosind operatorii pe biți).

4. Definiţi o structură de date ce ocupă spațiu minim, potrivită pentru a memora informația dacă un angajat a adus sau nu în dosarul firmei următoarele acte:
   - Copie buletin;
   - Copie certificat căsătorie;
   - Copie diplomă licență;
   - Copie diplomă master;
   - Copie diplomă doctor;
   - Fișă de lichidare de la locul de muncă anterior;
   - Certificate de naștere copii – pentru deducere impozit.