Programare Procedurala

Laborator 5

1. Structuri de date

Reamintim (lab 3):

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

2. Uniuni *(union)*

O uniune este un tip special de date care permite stocarea diferitelor tipuri de date in aceeasi locatie de memorie. Poate avea mai multi membri, insa un singur membru poate contine o valoare la un moment dat. Uniunile ne permit sa utilizam in mod eficient aceeasi locatie de memorie in mai multe scopuri. Atentie cum sunt folosite.

Declararea unei uniuni este similara 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 (fara **`nume_generic`**).
2. Pentru a declara variabile de tip **`nume_generic`** folosim constructia:

   ```
   union nume_generic variabila_union;
   ```

   sau adaugam numele variabilelor separate prin virgula inainte de punctul si virgula finala (in **`lista_variabile`**).
3. Rulați următorul exemplu:

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

   scanf("%d",&z.nr);
   printf("%d     %ld",z.nr,z.nr);
   ```
4. Dimensiunea unei uniuni va fi suficient de mare cat 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. Rulati urmatoarele instructiuni

```
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:

1. 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!

2. 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);
      }
```

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

4. Memoria ocupata: cat pentru **int.**

5. 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];
```

Observații:

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 biţi cu modificatorul `unsigned` va reţine valori între 0 și 7. Dacă nu apare modificatorul `unsigned`, atunci câmpul este cu semn şi va reţine 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 camp al structurii pentru care avem specificat numarul de biți:

```
printf("%d",&c1.pret);  // e ok;
printf("%d",&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));
```

Observații:

- **Verificați memoria ocupată!**
- Fara a utiliza campuri pe biti, ar fi fost necesari 8 octeti.

4. Accesarea membrilor este aceeasi 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 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!

Probleme

1. Sa se implementeze o functie care foloseste o uniune pentru a inversa cei doi octeti ai unui intreg (reprezentat pe 2 octeti) citit de la tastatura. Programul principal va apela functia pentru a codifica si decodifica un intreg dat.
   Exemplu: n = 20 -> 20 codificat este 5120
   5120 decodificat este 20

2. Folosind o singura structura, numita locuinta, memorati urmatoarele date:
   - adresa (cel mult 100 de caractere);
   - suprafata;

- tip locuinta (sir de cel mult 30 caractere): “garsoniera”, “casa” sau “apartament”;
- nr camere;
- in functie de tipul de locuinta, sa retinem:
  - pentru garsoniera: balcon/nu (1/0);
  - apartament: decomandat/nedecomandat (D/N);
  - casa: sir de caractere - una din variantele: “pe sol”, “parter+mansarda”, “nr etaje”;

Cerinte:

a. Cititi datele a **n** locuinte;
b. Afisati adresa garsonierei ce are balcon si totodata cea mai mare suprafata.

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 barbatii din firmă, mai tineri de 31 de ani (verificați vârsta folosind operatorii pe biți).

4.Definiţi o structură de date ce ocupa spatiu 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.