Programare Procedurala

Laborator 5

1. Uniuni *(union)*

O uniune este un tip special de date care permite stocarea diferitelor tipuri de date in aceeasi locatie de memorie. O uniune poate avea mai multi membri, insa un singur membru poate contine o valoare la un moment dat.

Declararea unei uniuni este similara cu cea a unei structuri:

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

- Putem avea uniuni anonime (fara **nume_generic**).
- 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).**

- Dimensiunea unei uniuni va fi suficient de mare cat cel mai mare membru al ei.

  Exemplu:

```
union alfa {
      char ch[3];
      int y;
} beta;
```

In exemplul de mai sus, dimensiunea tipului **alfa** este de 3 octeti, suficient cat sa tina vectorul de caractere **ch**. Verificati utilizand instructiunea de mai jos:

```
printf("Memoria ocupata de beta este de %d octeti", sizeof(beta));
```

- Uniunile ne permit sa utilizam in mod eficient aceeasi locatie de memorie in mai multe scopuri.

```
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;
}
```

➢ 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];
```

➢ 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;
    }
```

Programul va afisa : 0 1 2 3 4 5 6

- 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,   //0
    rosu = 3,  //3
    verde,   //4
    albastru = 8,  //8
    negru  //9
} culoare;
```

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

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];
```

- Daca un camp de biti este specificat ca **int** sau **unsigned,** atunci bitul cel mai semnificativ este bitul de semn
- Intr-o structura pot exista atat campuri pe biti, cat si membri normali
  Exemplu:

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

In exemplul de mai sus, toate informatiile sunt stocate intr-un singur octet (4 + 1 + 1 + 2). Fara a utiliza campuri pe biti, ar fi fost necesari cel putin 8 octeti.

- Accesarea membrilor este aceeasi ca in cazul structurilor:

```
grozavesti[0].camera = 10;
grozavesti[0].platit = 1;
grozavesti[0].ocupat = 1;
grozavesti[0].perioada_inchiriere = 2;
```

- 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);
//camera 0
```