STRUCTURI

Aspecte teoretice

Există situaţii în care tipurile de date învăţate până în prezent nu ne sunt de mare folos.
Presupunem că dorim să prelucrăm date referitoare la mai multi elevi.
Astfel, pentru fiecare elev cunoaştem:

1. Numele – char [20];
2. Prenumele – char[20];
3. Media la matematică – float;
4. Media la informatică –float;
5. Vârsta – int;

Observăm că informaţiile referitoare la un elev sunt de tipuri eterogene: şiruri de caractere, numere reale sau întregi. Cum am putea rezolva problema cu ajutorul cunoştinţelor de care dispunem? Ar fi necesari 5 vectori, câte unul pentru fiecare informaţie. O astfel de abordare este greoaie şi inflexibilă.
Este mult mai util să folosim un tip de dată prin care fiecărui elev să-i corespundă o singură înregistrare.

Soluţia C++ constă în folosirea tipului de date **struct**, care permite gruparea sub aceeaşi denumire a mai multor variabile (numite câpuri) de tipuri diferite, cât şi accesul şi operarea cu aceste câmpuri. Tipul **struct** mai poartă şi denumirea de tip de înregistrare.

O structură este compusă dintr-un număr de componente de anumite tipuri. Componentele structurii se numesc **câmpuri**.

Fiecare câmp trebuie să aparţină unui tip de date **deja definit** sau unui **tip standard.**

A. 1. Sintaxa declarării unei structuri:

```
struct  [nume_structura] {
                        tip_1    camp_1;
                        tip_2    camp_2;
                        ……….
                        tip_n    camp_n;

                    } [ lista_variabile_structura ];
```

Exemplul 1: Se defineşte structura elev care are 5 câmpuri

```
struct Elev {
   char nume [20];
   char prenume [20];
   float media_mate;
   float media_info;
   int varsta;
            };
```

```
struct Elev {
   char nume [20],  prenume [20];
   float media_mate, media_info;
   int varsta;
            };
```

Obs: Cele două structuri sunt echivalente, câmpurile care au acelaşi tip de date pot fi declarate în cadrul aceleiaşi declarări de tip, separate prin virgulă.

A.2 Există două posibilităţi de declarare a variabilelor care alcătuiesc structura:

1. Scriind la sfârşit numele variabilelor:

```
struct Elev
{ char nume[20], prenume[20];
  float media_mate, media_info;
 int varsta;
} e1,e2;
```

2. Declarând variabilele aşa cum suntem obişnuiţi:

```
Elev e1, e2;
```

Definiţia structurii poate fi făcută:

- În cadrul funcţiei main()
- Înaintea funcţiei main() (caz recomandat)

OBS: Numele structurii şi lista variabilelor pot lipsi dar nu simultan.
În cazul în care numele structurii lipseşte spunem că variabilele au **tip structură anonim**.

Exemple 2:

struct {
char nume [20];
char prenume [20];
float media;
} e1, e2, …, en;
	struct **Elev**{
	char nume [20];
	char prenume [20];
	float media;
	};
	
	Elev e1, e2, …, en;
		struct **Elev**{
		char nume [20];
		char prenume [20];
		float media;
		};
		
		struct Elev e1, e2, …, en;

A.3. Sintaxa de definire a unui tip structură:

```
typedef struct  {
                    tip_1    camp_1;
                    tip_2    camp_2;
                    ……..
                    tip_n    camp_n;

                } nume_tip;
```

Exemplu 3:

```
typedef struct {
                    char nume [20];
                    char prenume [20];
                    float media;
                    }Elev;
Elev e1, e2, e3;
```

A.4. Accesarea unui câmp al unei variabile ***a*** de tip structură, se face astfel:

```
a.camp_1;
a.camp_2;
……..
```

Pentru accesul la câmpurile unei variabile de tip struct se foloseşte operatorul de selecţie directă, notat cu '**.**', operator cu prioritate maximă.
Dacă *e* este o variabilă de tipul Elev atunci:

- ***e.nume*** – reprezintă şirul nume al variabilei inr;
- ***e.nume[0]*** - reprezintă primul caracter al şirului nume;
- ***e.nota_mate*** – reprezintă câmpul nota_mate al variabilei inr.

Între două variabile de acelaşi tip struct se poate folosi atribuirea.
Dacă e1 şi e2 sunt două variabile de tip elev, prin atribuirea e1=ie2, variabila e1 ia aceleaşi valori ca variabila e2. **O astfel de atribuire se mai numeşte copiere bit cu bit.**

Exemplu 4:

```
struct Elev{
                char nume [20];
                 char prenume [20];
                 float media;
                 } ;
Elev S;
```

Accesul variabilei de tip struct S la câmpurile nume, prenume, medie se face astfel:

S.nume;
S.prenume;
S.media;

B. 1. Citirea unei variabile de tip structură:

```
struct   nume_structura{
                              tip_1    camp_1;
                              tip_2    camp_2;
                              ……
                              tip_n    camp_n;
                            } a;
```

Metoda 1:	**Metoda 2:**
cin>>a.camp_1; **cin>>a.camp_2;** **…….** **cin>>a.camp_n;**	**cin>>a.camp_1>>a.camp_2>>...>>a.camp_n;**

Exemplu 5:

```
struct Elev{
          char nume [20];
          char prenume [20];
          float media;
          };
Elev e;
```

Metoda 1:	**Metoda 2:**

void citire_elev(Elev &e)
{
cout<<"Nume:" ; cin>>e.nume;
cout<<"Prenume:" ; cin>>e.prenume;
cout<<"Media:" ; cin>>e.media;
}
	void citire_elev(Elev &e)
	{
	cin>>e.nume>>e.prenume>>e.media;
	}

B.2. Afisarea unei variabile de tip structura:

```
struct   nume_structura{
                              tip_1    camp_1;
                              tip_2    camp_2;
                              ……
                              tip_n    camp_n;
                          } a;
```

Metoda 1:	**Metoda 2:**
cout<<a.camp_1<<endl; **cout <<a.camp_2<<endl;** **…….** **cout <<a.camp_n<<endl;**	**cout<<a.camp_1<<" "<<a.camp_2<<" "...<<a.camp_n;**

Exemplu 6:

struct **Elev**{
char nume [30];
char prenume [30];
float media;
};
Elev a;

Metoda 1:	**Metoda 2:**

void afisare_elev(Elev e)
{
cout<<"Nume:" ; cout<<e.nume<<endl;
cout<<"Prenume:" ; cout<<e.prenume<<endl;
cout<<"Media:" ; cout<<e.media<<endl;
}
	void afisare_elev(Elev e)
	{
	cout<<e.nume<<" "<<e.prenume<<" "<<e.media;
	}

Programul complet cu funcţiile de citire şi afişare a datelor unui elev este următorul:

```
1.  #include <iostream>
2.  using namespace std;
3.
4.  struct Elev
5.  { char nume [20];
6.    char prenume [20];
7.    float media;
8.  };
9.
10. void citire_elev(Elev &e)
11. {
12.   cout<<"Nume: " ; cin>>e.nume;
13.   cout<<"Prenume: " ; cin>>e.prenume;
14.   cout<<"Media: " ; cin>>e.media;
15. }
16.
17. void afisare_elev(Elev e)
18. {
19.   cout<<"Nume: " ; cout<<e.nume<<endl;
20.   cout<<"Prenume: " ; cout<<e.prenume<<endl;
21.   cout<<"Media: " ; cout<<e.media<<endl;
22. }
23.
24. int main()
25. {
26.   Elev e;
27.   citire_elev(e);
28.   afisare_elev(e);
29.   return 0;
30. }
```

C. Structuri imbricate

OBSERVAŢIE: Este posibilă definirea unei structuri ale cărei componente sunt de tipul structură, definite anterior. Câmpurile de tip structură se numesc structuri imbricate (incluse).

Exemplu 7:

```
struct Adresa{
                char strada [20];
                 char cartier [20];
                 int nr;
                };

struct Elev {
                char nume [20];
                 float media;
                 Adresa detalii;
                 }Elev;
Elev S;
```

Accesarea componentelor variabilei S se face astfel:

S.nume
S.detalii.strada
S.detalii.cartier
S.detalii.nr

D. Vectori de structuri (înregistrări)

Se poate declara un tablou cu elemente de tip structură.

Exemplu 8:

```
struct Elev {
                char nume [20];
                 char prenume [20];
                 float media;
                 }Elev;
Elev a[20];
```

D.1. Citirea a "n" componente ale vectorului "a" se poate face astfel:

Metoda 1:

```
cout<<"n="; cin>>n;
for (i=1; i<=n; i++)
  {
       cout<<"Nume:" ;       cin>>a[i].nume;
       cout<<"Prenume:" ;  cin>>a[i].prenume;
       cout<<"Media:" ;      cin>>a[i].media;
  }
```

Metoda 2:

```
cout<<"n="; cin>>n;
for (i=1; i<=n; i++)
  cin>>a[i].nume>>a[i].prenume>>...>>a[i].media;
```

D.2. Afisarea a “n” componente ale vectorului “a” se poate face astfel:

Metoda 1:

```
cout<<”n=”; cin>>n;
for (i=1; i<=n; i++)
  {
    cout<<”Nume:” ;       cout<<a[i].nume<<endl;
    cout<<”Prenume:” ;  cout<<a[i].prenume <<endl;
    cout<<”Media:” ;      cout<<a[i].media<<endl;
  }
```

Metoda 2:

```
for (i=1; i<=n; i++)
    cout<<a[i].nume<<” “<<a[i].prenume<<” “<<a[i].media<<endl;
```

Programul complet care citeşte şi afişează lista elevilor este prezentat în continuare:

```
1.  #include <iostream>
2.  using namespace std;
3.
4.  struct Elev
5.  { char nume [20];
6.    char prenume [20];
7.    float media;
8.  } ;
9.
10. void citire_elevi(Elev e[],int &n)
11. {
12.   int i;
13.   cout<<"n="; cin>>n;
14.   for(i=1;i<=n;i++)
15.   {
16.    cout<<"Nume: " ; cin>>e[i].nume;
17.    cout<<"Prenume: " ; cin>>e[i].prenume;
18.    cout<<"Media: " ; cin>>e[i].media;
19.   }
20. }
21.
22. void afisare_elevi(Elev e[],int n)
23. {
24.   int i;
25.   for(i=1;i<=n;i++)
26.   {
27.    cout<<"Nume: " ; cout<<e[i].nume<<endl;
28.    cout<<"Prenume: " ; cout<<e[i].prenume<<endl;
29.    cout<<"Media: " ; cout<<e[i].media<<endl;
30.   }
31. }
32.
33. int main()
34. {
35.   Elev e[31];
36.   int n;
37.   citire_elevi(e,n);
38.   afisare_elevi(e,n);
39.   return 0;
40. }
```

Asocierea unui nume pentru un tip de dată

Tipurile predefinite ale limbajului (de exemplu `int, char, float`) se identifică printr-un nume. După cum am văzut şi la definirea unui tip `struct` îi putem atribui un nume.

De fapt, programatorul poate asocia un nume oricărui tip de date, indiferent dacă acesta este un tip predefinit sau definit de el, utlizând instrucţiunea `typedef`:

```
typedef descriere_tip Nume_tip;
```

Se recomandă ca numele tipului de date nou `Nume_tip` să fie scris cu prima literă mare, astfel încât poate să fie foarte uşor diferenţiat de un tip standard (predefinit). Numele asociat şi descrierea tipului sunt sinonime,

Exemple 9:

1. Să asociem tipului predefinit float numele Real şi declarăm variabila r:
```
typedef float Real;
Real r;
```

2. Să asociem tipului `char *` (pointer la tipul de bază `char`) numele `Pchar` şi declarăm variabila p:
```
typedef char * Pchar;
Pchar p;
```

3. Să asociem numele `Produs` tipului întrgistrare care conţine descrierea unui produs şi declarăm var p:
```
typedef struct
{ char culoare[10];
  float greutate;
  float pret;
} Produs;
Produs p;
```

4. Să asociem numele `Vector` unui tablou unidimensional de componente de tip `int`
(nu se recomandă folosirea typedef în acest caz):
```
typedef int Vector[101];
Vector v;
```