Structuri

Pentru a organiza datele unui program limbajul pune la dispoziție pe de o parte tipurile de date simple (`int`, `double` etc) precum și tipuri de date compuse. Din aceasta categorie am studiat tablourile (*matricele* și *vectorii*). Acestea ne permit să declarăm o variabilă compusă formată din mai multe alte variabile componente, toate însă de același tip.

Sunt multe situații când dorim să ținem împreună informații care caracterizează un obiect și acestea să poată fi de tipuri diferite. De exemplu despre un produs dintr-un magazin e nevoie să memorăm informații precum: cantitate (câte unități de produs sunt în magazin – acesta este un număr întreg), preț unitar (cât costă o unitate de produs – acesta este un număr real) precum și numele (vom vedea mai târziu că pe acesta îl putem memora ca pe un șir de caractere).

Așadar ar fi necesară o variabilă compusă (o singură variabilă pentru a putea ține cumva împreună datele despre același lucru) în care componentele sale pot fi de tipuri diferite: un `int`, un `double` și un șir de caractere.

Limbajul oferă posibilitatea folosirii acestui fel de variabile dacă anterior definim tipul de date corespunzător. Aceste tipuri de date se numesc **structuri**.

Deci, mai întâi construim tipul de date nou și apoi vom declara variabile de noul tip.

În legătură cu structurile trebuie precizat că sunt mici diferențe între ceea ce permite compilatorul de C și cel de C++. Noi vom prezenta lucrurile așa cum sunt corecte sintactic dacă folosim un compilator de C++.

Definirea unui tip de date structură.

```
struct nume_tip_nou{
    tip_camp1 nume_camp1;
    tip_camp1 nume_camp1;
    …
    tip_campN nume_campN;
};
```

Iată și un exemplu pentru a ne fi mai ușor să clarificăm sintaxa referindu-ne la el.

```
struct produs{
    int cantitate;
    double pret;
};
```

`nume_tip_nou` (`produs` – în exemplu) reprezintă numele noului tip de date. Noi trebuie să ne gândim că într-un program ce conține această definire, după aceasta, noi dispunem de încă un tip de date, numit `produs`, pe lângă cele existente deja (`short`, `int`, `double` etc). Așadar ulterior vom putea declara variabile de noul tip.

`tip_camp1 nume_camp1 (int cantitate` – în exemplu) observăm că este ca o declarație de variabilă. În acest fel noi definim variabilele mai mici din care este formată cea compusă și acestea se numesc **câmpuri**. Altfel spus, când vom declara o variabilă de tip `produs`, aceasta va fi compusă din două simple, una de tip `int` numită cantitate și una de tip `double` numită pret.

O altă observație este că numele tipului de date (`produs`) și cele ale câmpurilor (`cantitate`, `pret`) respectă regula de numire a identificatorilor în limbaj (litere mari, litere mici, cifre, underline, dar să nu înceapă cu o cifră).

Cuvântul cheie `struct` este cel care dă semn compilatorului că este vorba despre un tip de date structură, apoi parantezele acolade sunt obligatorii și vor încadra declarațiile câmpurilor iar caracterul *punct și virgulă* de la final încheie definiția noului tip de date, fiind și el obligatoriu, parte din sintaxă.

Odată definit noul tip de date putem declara variabile folosind absolut aceeași sintaxă ca în cazul tipurilor de date predefinite:

```
tip_de_date nume_variabilă;
```

sau

```
produs p;
produs x, y;
```

Deasupra avem declarate trei variabile de tip `produs`. Pe prima am declarat-o independent iar pe celellte doua în același timp, separate prin virgulă, ca de obicei.

Declarațiile de tip `produs` de mai sus ne rezervă practic spațiu pentru trei zone de tip `int` și pentru trei de tip `double`, câte o zonă de tip `int` ținându-se împreună cu una de tip `double`, așa fiind construit noul tip de date.

Odată declarate variabile de tip structură, accesul la câmpuri se face prin construcții de forma:

```
nume_variabilă.nume_camp
```

Așadar mai sus dispunem de următoarele vatriabile de tip `int`: `p.cantitate`, `x.cantitate` și `y.cantitate` precum și de variabilele de tip `double`: `p.pret`, `x.pret` și `y.pret`.

Câmpurile sunt ca orice alte variabile simple de tipul lor putând fi prezente în construcții unde sunt permise variabile de acele tipuri.

Iată un tabel în care am strâns câteva astfel de exemple, în contextul definițiilor și declarațiilor de mai sus.

`p.cantitate = 20;`	Este vorba despre o atribuire către o variabilă de tip `int`, lucru permis.
`cout<<p.cantitate * 2;`	Se realizează o înmulțire între doi întregi (câmpul de tip `int` al lui `p` și constanta `2`) și rezultatul se afișează pe ecran.
`int r;` `r = p.pret % 2;`	Aici apare eroare de sintaxă din motiv că avem tipul `double` la câmpul de preț al lui `p` și operatorul `%` nu se poate aplica la date de tip `double`.
`produs.cantitate = 10;`	Și aici avem eroare de sintaxă pentru că nu este permisă accesarea unui câmp legată de tipul de date. Câmpul se poate accesa legat de o variabilă. Tragem concluzia că definirea unui tip de date structură care nu este urmată de declararea de variabile de noul tip nu implică alocarea de memorie.

```
#include <iostream>
using namespace std;
int a, b;
int main () {
    cin>>a>>b;
    cout<<a+b;
```

```
#include <iostream>
using namespace std;
int a[2];
int main () {
    cin>>a[0]>>a[1];
    cout<<a[0]+a[1];
```

```
#include <iostream>
using namespace std;
struct pereche {
    int a;
    int b;
};
```

}
 	
	}
	
		pereche x;
		int main () {
		  cin>>x.a>>x.b;
		  cout<<x.a+x.b;
		}

Cele trei programe de mai sus sunt echivalente din punct de vedere al funcționalității dar și din punct de vedere al gestionării memoriei. Primul folosește două variabile de tip `int` declarate independent. Al doilea, de asemenea două variabile de tip `int`, ca și componente ale unui vector de două `int`-uri, declarat anterior. În al treilea caz variabila `x` este de tip structură cu două câmpuri de tip `int`.

Operații cu structuri

Tipul de date structură nu vine implicit cu un set de operații prea bogat. Asta nu înseamnă că trebuie să tratăm cu dezinteres acest element de limbaj, ba dimpotrivă. Cele prezentate în această lecție reprezină baza programării orientate pe obiecte și în acest context putem îmbogăți noul tip de date cu orice operații dorim.

Singura operație cu structuri (dintre cele învățate până acum) pe care o putem folosi este **atribuirea**.

Adică, în contextul declarațiilor de mai sus (ne referim la tipul de date `produs`, cu două câmpuri: `cantitate` și `pret` și la variabile `x` și `y` de tip `produs`), este permisă o atribuire de forma:

```
x = y;
```

Această atribuire este echivalentă cu:

```
x.cantitate = y.cantitate;
y.pret = y.pret;
```

Alte operații dinte cele posibile la tipurile simple provoacă erori de compilare dacă se aplică structurilor. Iată câteva astfel de situații în care compilatorul cere corectarea codului:

`cin>>x;`	Nu este permis, dar am văzut că este posibilă citirea la nivel de câmp dacă acel câmp este de tipul unei variabile ce poate apărea parametru la `cin`.
`cout<<x;`	Aceeași situație.
`x + 2`	Aceeași situație.

Sunt permise diverse variante de combinare a definirii tipului de date cu declararea variabilelor.

Declararea de variabile chiar în momentul definirii tipului de date.

```
struct fractie {
    int numarator;
    int numitor;
} a, b;
fractie x, y;
```

Variabilele `x` și `y` au fost declarate exact ca în modul folosit în exemplele anterioare, după încheierea definirii tipului de date. Variabilele `a` și `b` au fost declarate în momentul definirii tipului de date, după acolada ce închide blocul de declarații dar înainte de caracterul *punct și virgulă*. Ele sunt, ca și celelalte două variabile, de tip `fractie`. Așadar sunt permise atribuiri de forma: `a = x; a = b; y = a;`

Declararea de variabile în momentul definirii structurii, de tip anonim.

```
struct {
    int numarator;
    int numitor;
```

```
} a, b;
```

Observați că aici s-a omis cuvântul ce indică numele noului tip de date, acolada deschisă fiind pusă imediat după cuvântul cheie `struct`. Este permisă această definire urmată însă imediat de declararea de variabile. În acest caz variabilele nu pot fi declarate decât imediat în momentul definirii, cum sunt `a` și `b`. Practic în acest caz se stabilește în spate un nume pentru noul tip iar `a` și `b` sunt de acel tip. Așadar noi putem ulterior să facem atribuiri de forma `a = b;`.

Pentru a înțelege mai bine ce se întâmplă, analizăm și următorul exemplu.

```
struct {
    int numarator;
    int numitor;
} a, b;

struct {
    int numarator;
    int numitor;
} c, d;
```

Cele două definiții de tip de date de mai sus sunt identice, ambele anonime. Asta nu înseamnă că `a` și `c`, spre exemplu, sunt de același tip. Este vorba despre două definiri anonime așadar se va aloca automat un nume pentru primul tip și alt nume pentru al doilea tip. Deci `a` și `b` sunt de același tip, fiind posibilă o atribuire de forma `a = b`, de asemenea `c` și `d` sunt ambele de același tip, fiind posibilă o atribuire de genul `c = d`, însă `a` și `c` sunt de tipuri diferite, o linie de forma `a = c` ducând la eroare de compilare.

Câmpuri de tip structură (anonimă sau nu) pentru o altă structură

Să considerăm următoarea secvență de cod:

```
struct data_nasterii {
    int ziua;
    int luna;
    int anul;
};
struct persoana {
    int cod;
    double salariu;
    data_nasterii d;
};
persoana a, b;
```

O variabilă de tip `persoana` (precum `a`) putem spune că are în componență un câmp de tip `int`, unul de tip `double` dar și unul de tip `data_nasterii`. Ținând cont că la câmpul `d`, de tip `data_nasterii`, se ascund încă trei `int`-uri, putem spune că avem pentru o persoană patru câmpuri de tip `int` și unul de tip `double`.

Cum accesăm câmpurile lui `a`?

`a.cod` este un câmp de tip `int`

`a.salariu` este câmpul de tip `double`

`a.d.ziua` este câmpul în care memorăm ziua din data nașterii. Nu putem scrie direct `a.ziua` ci trecem de la fiecare structură direct în câmpurile ei. Adică `d` este câmp de tip structură al lui `a`, scriem deci `a.d` iar din `d` putem scrie în continure *punct* urmat de câmpurile sale.

Dacă nu ar fi nevoie să declarăm separat și variabile de tip `data_nasterii`, cele de mai sus puteau fi scrise și:

```
struct persoana {
    int cod;
    double salariu;
    struct {
        int ziua;
        int luna;
        int anul;
    } d;
};
persoana a, b;
```

Aici am omis să mai numim tipul `data_nasterii` și am scris câmpurile din `persoana` de tip anonim.

Sau

```
struct {
    int cod;
    double salariu;
    struct {
        int ziua;
        int luna;
        int anul;
    } d;
} a, b;
```

Aici am omis numim ambele tipuri structură.

Recomandarea este de a folosi la dezvoltarea aplicațiilor o separare a lucrurilor, adică așa cum am lucrat la început: definesc noul tip de date, numindu-l, și apoi, de câte ori doresc, declar variabile de noul tip.

Folosirea aceluiași identificator și pe post de nume de câmp și pe post de nume de variabilă.

```
struct punct {
    int x;
    int y;
};
punct x;
```

Aici compilatorul nu sesizează dublare a numelui de identificator (`x`) pentru că un `x` este în interiorul blocului de acolade. Lucrurile sunt în regulă, `x.x` și `x.y` fiind ambele variabile de tip `int`.

Declararea mai multor câmpuri de același tip în același timp.

Ne referim la faptul că tipul de date din exemplul de mai sus ar putea fi definit și ca mai jos:

```
struct punct {
    int x, y;
};
```

Structuri cu câmpuri de tip vector

Orice tip de date poate fi folosit pentru a defini un câmp al unei structuri. Putem avea chiar câmpuri de tip tablou la o structură. Mai jos vom defini cele necesare memorării situatiei semestriale pentru un elev la o disciplină la care se susține teza.

```
struct situatie {
    int numarnote;
    int note[10];
    int teza;
};
situatie mate, info;
```

Programul de mai jos cere de la tastatură numărul de note, notele și nota la teză ale unui elev la disciplina informatică și afișează pe ecran media sa.

```
int main () {
    cin>>info.numarnote;
    for (int i=0;i<info.numarnote;i++)
        cin>>info.note[i];
    cin>>info.teza;
    double suma = 0;
    for (int i=0;i<info.numarnote;i++)
        suma += info.note[i];
    cout<<(int)((suma / info.numarnote + info.teza)/2.0);
}
```

În program am considerat că teza are ponderea a jumătate din medie.

Respectând logica lucrurilor: `info` este o variabilă de tip structură iar `note` un câmp al său, deci scriem `info.note`. Acesta fiind un câmp de tip tablou unidimensional, elementele sale le accesăm folosind parantezele drepte. De aici ajungem la sintaxa folosită: `info.note[indice]`.

Vector de structuri

Avem ca task să memorăm mai multe puncte din plan (pentru fiecare coordonata `x` și coordonata `y`) apoi să le sortăm crescător după abscisă (`x`) și în caz de egalitate după ordonată (`y`).

Vom defini o structură cu două câmpuri de tip `int`, pentru a memora un punct. Întrucât avem de lucrat cu mai multe puncte vom folosi un vector de astfel de structuri.

```
#include <iostream>
using namespace std;
struct punct {
    int x;
    int y;
};
punct v[101];
punct aux;
int n, i, j;
int main () {
    cin>>n;
    for (i=1;i<=n;i++)
        cin>>v[i].x>>v[i].y;
    for (i=1;i<n;i++)
        for (j=i+1;j<=n;j++)
    if (v[i].x>v[j].x || (v[i].x==v[j].x && v[i].y>v[j].y )) {
        aux = v[i];
        v[i] = v[j];
        v[j] = aux;
    }
    for (i=1;i<=n;i++)
```

```
        cout<<v[i].x<<" "<<v[i].y<<"\n";
}
```

Așa cum spuneam, în afară de atribuire, alte operații cu structuri nu sunt permise. Deci noi la comparare testăm ordinea folosind operatorul relațional la nivel de câmp, dar la interschimbare utilizăm o variabilă de tip `punct` pentru a atribui deodată două structuri (echivalent cu a face atribuiri simple la nivel de câmp).

Probleme rezolvate

1. Se citesc de la tastatură numărătorul și numitorul pentru două fracții raționale, pozitive (se dau `4` numere naturale nenule cu maxim `4` cifre în ordinea: numărător1, numitor1, numărător2, numitor2). Să se afișeze pe ecran, sub formă ireductibilă, fracția produs și fracția sumă.

Exemplu

Date de intrare	
1 2 4 5	2 5 13 10

Rezolvare

Vom defini un tip structură, numit `fractie` și care are două câmpuri de tip `int`, numărătorul și numitorul. Folosim apoi `4` variabile de tip fracție: `f1`, `f2`, `produs` și `suma`. Pentru produs rezultatul se obține înmulțind numărătorii între ei și numitorii între ei. Pentru sumă, nu vom proceda întocmai ca la matematică găsind cel mai mic multiplu comun al numitorilor ci ne vom mulțumi cu orice multiplu comun al numitorilor, de exemplu produsul lor. Pentru aceasta vom amplifica prima fracție cu numitorul celei de-a doua și pe a doua cu numitorul primei. Fracțiile rezultat trebuie simplificate. Acest lucru se realizează determinând cel mai mare divizor comun dintre numărător și numitor și împărțind pe ambele la această valoare.

```
#include <iostream>
using namespace std;
struct fractie {
    int numarator;
    int numitor;
};
fractie f1, f2, produs, suma;
int a, b, r;
int main () {
    cin>>f1.numarator>>f1.numitor;
    cin>>f2.numarator>>f2.numitor;
    produs.numarator=f1.numarator*f2.numarator;
    produs.numitor=f1.numitor*f2.numitor;
    suma.numarator=f1.numarator*f2.numitor+f1.numitor*f2.numarator;
    suma.numitor=f1.numitor*f2.numitor;
    a = produs.numarator;
    b = produs.numitor;
    while (b) {
        r = a%b;
        a = b;
        b = r;
    }
    produs.numarator/=a;
    produs.numitor/=a;
    a = suma.numarator;
    b = suma.numitor;
```

```
    while (b) {
        r = a%b;
        a = b;
        b = r;
    }
    suma.numarator/=a;
    suma.numitor/=a;
    cout<<produs.numarator<<" "<<produs.numitor<<"\n";
    cout<<suma.numarator<<" "<<suma.numitor<<"\n";
    return 0;
}
```

2. Se citesc de la tastatură informații despre produsele dintr-un magazin: numărul de produse, apoi, pentru fiecare produs: codul, cantitatea, prețul unitar.

a) Memorați datele despre produsele citite;

b) Afișați toate informațiile despre produsul cu prețul unitar minim (se garantează că este un singur astfel de produs);

c) Afișați numărul de produse care au cea mai mare valoare totală în stoc;

Exemplu

Date de intrare	Date de ieșire	Explicație
3 1 4 2 2 3 4 3 2 6	1 4 2 2	Avem 3 produse. Primul are codul 1, se găsește în cantitate de 4 unități și prețul pe unitate este 2. La b) se vor afișa datele sale (are cel mai mic preț unitar). Celelalte două produse au valoarea totală 12 (al doilea: 3 (unități) * 4 (prețul pe unitate) iar la al treilea 2*6). Deoarece primul produs are valoarea totală 8, rezultatul este 2.

Rezolvare

Vom folosi o structură cu trei câmpuri, câte unul pentru fiecare informație despre produs. Datele se vor stoca într-un vector cu astfel de structuri. La cerința b) aflăm indicele din vector pentru elementul ce are câmpul preț unitar minim și apoi afișăm informațiile de la poziția determinată. Pentru ultima cerință avem soluția clasică de aflare a maximului și numărului de apariții (maximul între valori cantitate*preț unitar).

```
#include <iostream>
using namespace std;
struct produs {
    int cod;
    int cantitate;
    double pret;
};
int n, i, poz, sol, minim, maxim;
produs v[101];
int main () {
    cin>>n;
    for (i=1;i<=n;i++)
        cin>>v[i].cod>>v[i].cantitate>>v[i].pret;
    minim = v[1].pret;
    poz = 1;
    for (i=2;i<=n;i++)
        if (v[i].pret < minim) {
```

```
            minim = v[i].pret;
            poz = i;
        }
    cout<<v[poz].cod<<" "<<v[poz].cantitate<<" "<<v[poz].pret<<"\n";
    maxim = -1;
    for (i=1;i<=n;i++) {
        if (v[i].cantitate * v[i].pret > maxim) {
            maxim = v[i].cantitate * v[i].pret;
            sol = 1;
        } else
            if (v[i].cantitate * v[i].pret == maxim)
                sol++;
    }
    cout<<sol;
    return 0;
}
```

3. Se dau n numere naturale nenule. Ordonați-le descrescător după numărul lor de divizori. Dacă există mai multe numere cu același număr de divizori acestea se vor afișa în ordine crescătoare (pbinfo.ro, #1608).

Exemplu

Date de intrare	Date de iesire	Explicație
5 12 20 4 100 13	100 12 20 4 13	12 are 6 divizori, 20 are 6 divizori, 4 are 3 divizori, 100 are 9 divizori, 13 are 2 divizori, 12 și 20 au același număr de divizori. Așadar ordinea va fi 100 12 20 4 13.

Rezolvare

Noi pentru fiecare valoare dată avem de calculat numărul de divizori iar la sortare acesta va fi primul criteriu de comparare. Dacă am folosi doi vectori, unul cu numerele date și altul cu numărul de divizori în dreptul fiecăruia (pe aceeași poziție), ar trebui să realizăm interschimbarea în ambii vectori în același timp. Cel mai comod și natural este să definim o structură cu două câmpuri (semnificând valoarea elementului și numărul de divizori), ținând astfel împreună datele aceluiași element de la intrare. Atribuirea fiind permisă la structuri ne va fi mai ușor și la interschimbare.

```
#include <fstream>
using namespace std;
ifstream fin ( "sortare_divizori.in" );
ofstream fout ( "sortare_divizori.out" );
int  n, i, j, d, k, nrDiv, e;

struct element {
    int valoare;
    int divizori;
};
element a[1001], aux;
int main(){
    fin>>n;
    for (i=1;i<=n;i++){
        fin>>a[i].valoare;
        int t = a[i].valoare;
        d = 2;
```

```
        nrDiv = 1;
        while (d*d <= t && t!=1){
            e = 0;
            while (t % d == 0){
                e++;
                t/=d;
            }
            if (e != 0){
                nrDiv *= (e+1);
            }
            d++;
        }
        if (t!=1){
            nrDiv *= (1+1);
        }
        a[i].divizori = nrDiv;
    }
    for (i=1;i<n;i++)
        for (j=i+1;j<=n;j++)
            if ((a[i].divizori<a[j].divizori)||
                ((a[i].divizori==a[j].divizori)&&a[i].valoare>a[j].valoare)){
                aux = a[i];
                a[i] = a[j];
                a[j] = aux;
            }
    for (i=1;i<=n;i++)
        fout<<a[i].valoare<<" ";
}
```

Remarcați modul în care am ales să calculăm numărul de divizori ai unui număr: Dacă l-am factoriza (descompunere în factori primi), adică l-am scrie sub forma:

$$t = d_1^{e_1} \bullet d_2^{e_2} \cdot \ldots \cdot d_k^{e_k}$$

Atunci numărul de divizori ai lui `t` este: $(1+e_1)(1+e_2)\ldots(1+e_k)$. Poate că ne amintim de la matematică acest rezultat dar iată și un raționament simplu pe care să îl avem la îndemână pentru a ține minte ușor formula: fiecare factor prim apare în fiecare divizor la orice putere între `0` (adică factorul respectiv nu apare deoarece $p^0=1$) și `e`. Deci sunt `e+1` variante de a alege cum contribuie la un divizor un anume factor prim. Orice combinare de astfel de alegeri de la fiecare factor prim este un divizor. De aici rezulta formula de mai sus. Remarcați și optimizarea făcută pentru a căuta factori primi până la radicalul numărului și considerarea valorii rămase după radical la puterea `1` (dacă `t`, valoarea de descompus nu este deja `1`).

4. Se dau `n` numere naturale. Afișați cifrele care apar în scrierea zecimală a acestor numere, în ordinea crescătoare a numărului de apariții. Dacă două cifre au același număr de apariții, se va afișa mai întâi cifra mai mică.

Exemplu

Date de intrare	Date de ieșire
5 124 229 1322 4 534	5 9 1 3 4 2

Rezolvare

Vom folosi un vector de frecvență `f` pentru a calcula numărul de apariții pentru fiecare cifră. Apoi, construim un vector de structuri în care elementele au valori (`i`, `f[i]`) pentru elementele nenule din `f` (`v[k].cifra`

= i și v[k].aparitii = f[i]). Acum ne rămâne de lucrat doar cu acest vector (sortare după criteriile cerute și apoi afișare a câmpurilor cifra).

```
#include <iostream>
using namespace std;
struct element {
    int cifra;
    int aparitii;
};
int f[10];
int n,i,j,k, x;
element v[11], aux;
int main(){
    cin>>n;
    for (i=1;i<=n;i++) {
        cin>>x;
        if (x == 0)
            f[0]++;
        while (x!=0) {
            f[x%10]++;
            x /= 10;
        }
    }
    for (i=0;i<=9;i++)
        if (f[i] != 0) {
            /// cifra i a aparut de f[i] ori
            k++;
            v[k].cifra = i;
            v[k].aparitii = f[i];
        }
    for (i=1;i<k;i++)
        for (j=i+1;j<=k;j++)
            if (v[i].aparitii>v[j].aparitii ||
               (v[i].aparitii>v[j].aparitii && v[i].cifra>v[j].cifra))
{
                aux = v[i];
                v[i] = v[j];
                v[j] = aux;
            }
    for (i=1;i<=k;i++)
        cout<<v[i].cifra<<" ";
    return 0;
}
```