# TIPURI COMPLEXE DE DATE – STRUCTURI

O structură este o colecție de una sau mai multe variabile (de acelaşi tip sau de tipuri diferite) grupate sub un singur nume. Variabilele care fac parte din structură se numesc membrii structurii respective. Membrii unei structuri pot fi tipuri de date simple (int, float, etc), tipuri de date agregate (tablouri, alte tipuri de structuri) sau pointeri (la diverse tipuri de date, inclusiv pointer la acelaşi tip de structură – structuri autoreferite).

## 1. Declarația şi definiția unei structuri

### 1.1. Declarația unei structuri

Forma generală a declarației unei structuri este:

```
struct NUME_STRUCTURĂ {
        declarații de date;
};
```

şi cuprinde:

- Cuvântul cheie **struct**
- Numele modelului (prototipului) structurii **NUME_STRUCTURĂ**
- Descrierea membrilor structurii respective. Forma generală a membrilor unei structuri este
  - tip nume;

sau

  - tip nume[$dim_1$][$dim_2$]…[$dim_n$];

Prin această descriere se indică "prototipul" (modelul) structurii respective, nu se rezervă zonă de memorie şi această declarație poate apare într-un fişier header.

O structură se poate declara şi fără indicarea componenței sale, adică putem scrie:

```
struct NUME_STRUCTURĂ;
```

urmând ca descrierea să se facă ulterior, înainte de prima definire a unei variabile al cărui tip este structura respectivă.

Declarația unei structuri va apare în fişierul header înainte oricăror referiri la tipul de structură respectiv (de exemplu, înaintea specificării prototipurilor funcțiilor care folosesc ca parametru sau ca valoare de retur o structură de tipul specificat).

### 1.2. Definiția unei structuri

Are rolul de a rezerva zonă de memorie pentru variabilele de tip structură forma generală este:

```
struct [NUME_STRUCTURĂ] {
    declarații de date;
} v1, v2[10], …, vn;
```

unde v1, v2, …, vn sunt variabile de tipul NUME_STRUCTURĂ. Aici […] indică faptul că partea respectivă din definiție poate să lipsească.

În cazul în care structura a fost declarată anterior (în fişierul header), definirea unor variabile de tip structură se face în forma:

```
struct NUME_STRUCTURĂ v1, v2[10], …, vn;
```

Definiția unei structuri nu poate apare într-un fişier header.

## 2. Operații cu structuri

### 2.1. Inițializarea

Forma generală pentru inițializarea unei structuri este:

```
struct NUME_STRUCTURĂ variabila_structura = {val1, …, valn};
```

dacă structura a fost declarată anterior sau

```
struct NUME_STRUCTURĂ {
    declarații de date;
} variabila_structura = {val1, …, valn};
```

dacă se face şi definirea variabilei de tip structură o dată cu declararea ei. $val_1$, …, $val_n$ reprezintă valorile de inițializare ale membrilor structurii respective şi trebuie să coincidă cu tipul membrului structurii la care se referă.

Exemplu:

Fiind dată o declarație de structură care descrie o dată calendaristică de forma:

```
struct DATA {
    int zi;
    char luna[15];
    int an;
};
```

Definirea cu inițializare a unei variabile ziDeNastere de tip DATA se face prin:

```
struct DATA ziDeNastere = {11, "iunie", 1970};
```

## 2.2. Accesarea membrilor

Un membru al unei structuri se accesează folosind operatorul de selecție **.** (punct).

Forma generală este:

variabila_structura.membru_structura

Luând exemplul de mai sus, afişarea conținutului variabilei de tip structură **ziDeNaştere** se face astfel:

```
printf("%d %s %d\n", ziDeNastere.zi, ziDeNastere.luna, ziDeNastere.an);
```

Observație:

Atunci când un membru al unei structuri este de un tip real (float, double sau long double) inițializarea prin citire de la tastatură a valorii membrului respectiv se face folosind o variabilă auxiliară, ca în exemplul următor:

```
PUNCT a;
double aux;
scanf("%lf", &aux);
a.x = aux;
```

## 2.3. Copierea unei structuri

Este permisă copierea unei structuri în altă structură fie membru cu membru (prin atribuiri aplicate unor variabile simple), fie luând structura ca un întreg.

Exemplu:

```
struct DATA zi1 = {25, "martie", 2004};
struct DATA zi2, zi3;
zi2 = zi1;
zi3.zi = zi1.zi;
```

după secvența de cod de mai sus, membrii structurii **zi2** vor avea aceleaşi valori ca şi membrii structurii **zi1**, iar membrul **zi** al structurii **zi3** va avea aceeaşi valoare cu membrul **zi** al structurii **zi1**, ceilalți doi membri rămânând cu valori nedeterminate.

## 2.4. Structuri şi funcții

O structură poate fi folosită ca parametru de intrare pentru o funcție şi o funcție poate returna o structură. Modificările făcute asupra valorii membrilor unei structuri într-o funcție nu se păstrează şi după ieşirea din funcție. O structură se transmite către o funcție prin valoare, prin intermediul stivei.

Exemplu:

Fiind declarată structura următoare asociată unui punct din plan

```
struct  _PUNCT {
        double x, y;
};
typedef struct  _PUNCT PUNCT;
```

a) să se scrie o funcție care returnează distanța dintre două puncte;

b) să se scrie o funcție care returnează punctul simetric față de origine al unui punct dat.

a)
```
double distanta(PUNCT a, PUNCT b)
{
        double d;
        d = sqrt((a.x – b.x) * (a.x – b.x) + (a.y – b.y) * (a.y – b.y));
        return d;
}
```

b)
```
PUNCT simetric(PUNCT a)
{
        PUNCT b;
        b.x = -a.x;
        b.y = -a.y;
        return b;
}
```

Considerând următoarele definiții:

```
double d;
PUNCT A, B, X, Y;
```

şi că variabilele structură de tip PUNCT A, B, Y sunt inițializate pe parcursul programului, apelul celor două funcții se face astfel:

```
d = distanta(A, B);
/*
 *    Calculează distanța dintre punctele ale căror coordonate
 *        sunt în structurile A şi B de tip PUNCT
 */

X = simetric(Y);
/*
 *    Membrii variabilei structură X, de tip PUNCT, va conține valorile simetrice
 *        față de originea axelor de coordonate ale membrilor variabilei structură Y,
 *        de asemenea de tip PUNCT.
 */
```

# TEMA

**Pentru fiecare problemă se va construi un proiect cu fişierul header corespunzător.**

## Problema nr. 1

Se consideră următoarea structură asociată unui punct din spațiul bidimensional:

```
struct  _PUNCT {
        double x;       // abscisa
        double y;       // ordonata
};
```

şi declarația de tip

```
typedef struct  _PUNCT PUNCT;
```

Folosind aceste două declarații să se rezolve următoarea problemă:

Se citeşte dintr-un fişier un număr întreg **n** şi apoi coordonatele a **n** puncte din spațiul bidimensional (fiecare linie conține coordonatele unui punct separate printr-un spațiu). Exemplu: Fişierul de intrare este **poligon.in** şi are următorul conținut:

```
3
1 1
-2 4
-4 -3
```

Să se calculeze aria şi perimetrul poligonului determinat de cele **n** puncte, presupunând că poligonul este convex. Valorile pentru arie şi perimetru se vor afişa după afişarea coordonatelor vârfurilor poligonului cu 3 zecimale.

Se vor scrie şi folosi funcții pentru:

- Calculul distanței dintre două puncte;
- Calculul ariei unui triunghi folosind formula lui Heron;
- Calculul semiperimetrului unui triunghi;
- Calculul ariei totale;
- Calculul perimetrului poligonului
- Citirea unui vector de structuri dintr-un fişier.

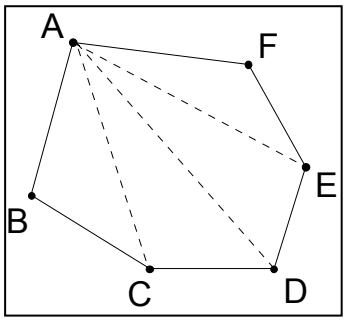
- Afişarea coordonatelor vârfurilor poligonului ştiind că primul vârf este notat cu **A**, iar coordonatele unui vârf se indică între paranteze după "numele" vârfului.

  Exemplu:

  A(1, 1) B(2.75, 3.14)

  Pentru coordonate de tip real se vor indica două zecimale şi se vor afişa câte 5 puncte pe o linie, puncte separate printr-un TAB.

Exemplu:




Perimetrul = ∑ lungimilor laturilor AB, BC, CD, DE, EF, FA

Aria = ∑ ariilor triunghiurilor ABC, ACD, ADE, AEF

Date de test:

a). pentru poligonul A(1,1), B(-2,4), C(-4,-3)

Aria = 13,500

Perimetrul = 17,926

b) pentru poligonul A(2,98; 8,88), B(9,02; 5,98), C(13,16; 10,12), D(11,02; 14,00), E(6,04; 18,08)

Aria = 68,732

Perimetrul = 33,119

## Problema nr. 2

În pădurea cu alune aveau case mai mulţi pitici. Unii dintre ei purtau scufii roşii, alţii purtau scufii albastre. Regele lor era singurul care purta scufie galbenă.

Din nefericire au apărut conflicte între piticii cu scufii roşii şi piticii cu scufii albastre. Pentru a-şi putea păstra scufia galbenă, regele trebuie să găsească o soluţie pentru a aplana acest conflict. El plănuieşte să construiască o potecă prin pădure care să separeu casele piticilor cu scufii roşii de casele piticilor cu scufii albastre. Pentru aceasta toţi sfetnicii trebuie să propună poziţii ale potecii. Se cunosc coordonatele tuturor căsuţelor piticilor şi fiecare sfetnic va propune coordonatele a două puncte care se vor afla pe dreapta care reprezintă poteca.

O potecă este validă dacă şi numai dacă toate casele piticilor cu scufii roşii se vor afla de o parte a potecii şi toate casele piticilor cu scufii albastre se vor afla de cealaltă parte. O potecă

nu va fi validă dacă va trece exact prin casa unui pitic.(indiferent de culoarea scufiei acestuia. Trebuie să stabiliţi care din potecile propuse de sfetnici sunt valide.

**Date de intrare**

Prima linie a fişierului de intrare **PITICI.IN** conţine numărul *M* al piticilor cu scufii roşii. Următoarele *M* linii conţin perechi de numere reale, separate printr-un spaţiu, reprezentând coordonatele căsuţei unui pitic cu scufie roşie.

Următoarea linie conţine numărul *N* al piticilor cu scufii albastre. Următoarele *N* linii conţin perechi de numere reale, separate printr-un spaţiu, reprezentând coordonatele căsuţei unui pitic cu scufie albastră.

Următoarea linie va conţine numărul *K* al sfetnicilor, iar pe următoarele *K* linii se vor afla câte patru numere reale, separate printr-un spaţiu, reprezentând coordonatele celor două puncte care vor determina dreapta propusă de sfetnic.

**Date de ieşire**

Fişierul de ieşire **PITICI.OUT** va conţine *K* linii fiecare corespunzând unei poteci propuse de un sfetnic. în cazul în care poteca este validă linia va conţine mesajul DA – Sfetnicul *i* a dat o soluţie valida, iar în cazul în care poteca nu este validă linia va conţine mesajul NU – Sfetnicul *i* a dat o soluţie invalida. În ambele cazuri *i* reprezintă numărul de ordine al sfetnicului care a dat soluţia respectivă.

**Exemplu**:

| PITICI.IN | PITICI.OUT |
|---|---|
| 2 | NU - Sfetnicul 1 a dat o solutie invalida |
| 0 0 | DA - Sfetnicul 2 a dat o solutie valida |
| 0 2 | NU - Sfetnicul 3 a dat o solutie invalida |
| 2 | NU - Sfetnicul 4 a dat o solutie invalida |
| 2 0 | |
| 2 2 | |
| 4 | |
| 0 1 2 1 | |
| 1 0 1 2 | |
| 0 0 1 2 | |
| 1 2 2 0 | |

*Indicaţie de rezolvare*:

Problema se reduce la a verifica pentru fiecare dintre poteci (drepte) dacă punctele corespunzătoare căsuţelor tuturor piticilor cu scufii roşii se află de o parte a dreptei şi punctele corespunzătoare căsuţelor tuturor piticilor cu scufii albastre se află de cealaltă parte a dreptei.

Pentru aceasta trebuie determinată ecuaţia dreptei corespunzătoare unei poteci. Ecuaţia unei drepte este:

$$ax + by + c = 0 \qquad (1)$$

Fie $(x_1, y_1)$ şi $(x_2, y_2)$ coordonatele a două puncte care determină o dreaptă. Coeficienţii din formula (1) se determină cu relaţiile:

$$a = y_1 - y_2 \qquad b = x_2 - x_1 \qquad c = x_1 y_2 - x_2 y_1 \tag{2}$$

Pentru a determina de care parte a dreptei (potecii) se află căsuţa unui pitic trebuie să înlocuim în formula (1) variabilele $x$ şi $y$ cu coordonatele căsuţei respective. Pentru o căsuţă care se află chiar pe potecă rezultatul va fi nul, iar pentru căsuţele piticilor cu scufii roşii rezultatul trebuie să aibă un semn, iar pentru căsuţele piticilor cu scufii albastre rezultatul trebuie să aibă semn opus.

Trebuie să determinăm semnul pentru căsuţa primului pitic cu scufie roşie. Apoi vom determina semnele pentru căsuţele tuturor celorlalţi pitici cu scufii roşii. Dacă apare un semn diferit putem trage imediat concluzia că poteca nu este validă. Dacă obţinem o valoare nulă dreapta corespunzătoare va trece prin căsuţa unui pitic, deci nici în această situaţie poteca nu este validă.

Dacă nu am identificat nici o situaţie care să conducă la concluzia că poteca nu este validă, putem fi siguri că toţi piticii cu scufii roşii au căsuţele de aceeaşi parte a potecii.

În continuare vom determina semnele pentru căsuţele piticilor cu scufii albastre. Dacă apare valoarea 0 sau acelaşi semn ca pentru piticii cu scufii roşii rezultă imediat că poteca nu este validă.

Dacă nu am identificat nici acum o situaţie care să conducă la concluzia că poteca nu este validă, putem fi siguri că toţi piticii cu scufii albastre au căsuţele de cealaltă parte a potecii.

în acest moment putem concluziona că poteca respectivă este validă.

Pentru rezolvarea problemei trebuie definite tipurile de date:

PUNCT ca sinonim pentru o structură asociată unui punct din plan care conţine ca membri două date de tip real;

PITICI ca sinonim pentru o structură asociată piticilor cu scufii de aceeaşi culoare. Structura cuprinde ca membri un întreg fără semn (reprezentând numărul piticilor) şi un tablou de structuri PUNCT prin intermediul căruia se vor stoca coordonatele căsuţelor piticilor din grupul respectiv.

După citirea coordonatelor căsuţelor celor două grupuri de pitici aceste se vor afişa pe ecran respectând următorul model (dat pentru exemplul din problemă):

Casutele piticilor cu scufii rosii se gasesc la urmatoarele coordonate:

(0.0, 0.0) (0.0, 2.0)

Casutele piticilor cu scufii albastre se gasesc la urmatoarele coordonate:

(2.0, 0.0) (2.0, 0.2)

Coordonatele unui punct se vor afişa cu o singură zecimală.

## Problema nr. 3

Se consideră polinomul cu coeficienți complecşi:

$$P(x) = a_0 \cdot x^n + a_1 \cdot x^{n-1} + \ldots + a_{n-1} \cdot x + a_n \qquad (2.1)$$

Să se calculeze, folosind schema lui Horner, valoarea polinomului în punctul **z** din planul complex, **z** fiind definit cu expresia

$$z = x + j \cdot y \qquad \text{unde } j = \sqrt{-1} \qquad (2.2)$$

Relația de recurență pentru calculul valorii unui polinom într-un punct **z** este:

$$P_n(z) = P_{n-1}(z) \cdot z + a_n \quad \text{cu} \quad P_{-1}(z) = 0 \qquad (2.3)$$

(nu se vor folosi funcții recursive).

Se asociază unui număr complex structura:

```
struct COMPLEX {
    double re;      // partea reală
    double im;      // partea imaginară
};
```

şi unui polinom structura:

```
struct POLINOM {
    int n;                      // gradul polinomului
    struct COMPLEX c[20];       // coeficienții polinomului
};
```

Se vor scrie şi folosi funcții pentru:

- Citirea unui vector de numere complexe dintr-un fişier (**polinom.in**);
- Adunarea a două numere complexe;
- Înmulțirea a două numere complexe;
- Calculul valorii unui polinom de variabilă complexă cu coeficienți complecşi (varianta iterativă şi nu recursivă) într-un punct.

## Problema nr. 4

Fie tipul de dată PUNCT declarat astfel:

```
typedef struct  _PUNCT {
        double x;       // abscisa
        double y;       // ordonata
} PUNCT;
```

asociat unui punct din plan şi un tip de dată declarat:

```
typedef struct  _LINIE {
        double m;       // panta
        double n;       // tăietura
} LINIE;
```

asociat unei drepte a cărei ecuație este:

$$y = m \cdot x + n \tag{3.1}$$

Se citeşte, de la tastatură, un număr întreg **n** şi apoi coordonatele a **n** puncte din plan.

a) Să se verifice dacă toate punctele se găsesc pe dreapta determinată de primele două puncte introduse. Pentru aceasta se determină ecuația unei drepte luând în considerare coordonatele primelor două puncte din cele introduse şi apoi se verifică dacă celelalte puncte aparțin dreptei.

Ecuația dreptei care trece prin două puncte este:

$$\frac{y - y_1}{y_2 - y_1} = \frac{x - x_1}{x_2 - x_1} \tag{3.2}$$

unde $(x_1, y_1)$ şi $(x_2, y_2)$ sunt coordonatele punctelor care determină dreapta.

b) Să se afişeze ecuația dreptei dată de ecuația (3.1).

c) Să se afişeze coordonatele punctelor care nu sunt pe dreapta care trece prin primele două puncte din plan ale căror coordonate au fost introduse. Afişarea coordonatelor punctelor care nu sunt pe dreaptă se face sub forma **(x, y)**.

Se vor scrie şi folosi funcții pentru:

- Citirea unui vector de structuri

- Determinarea ecuației unei dreptei (determinarea pantei şi tăieturii dreptei respective)
- Verificarea coliniarității punctelor introduse
- Afişare ecuație dreaptă
- Determinare puncte care nu sunt pe dreaptă
- Afişare puncte care nu sunt pe dreaptă

Observație:

Trei puncte sunt coliniare dacă

$$\begin{vmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{vmatrix} = 0$$

unde ($x_i$, $y_i$) cu i = 1, 2, 3 sunt coordonatele celor trei puncte. Relația de mai sus poate fi folosită (cu adaptările necesare) şi pentru determinarea ecuației unei drepte care trece prin două puncte.