

Structuri

Cuprins:

Noțiuni Generale.....	1
Bune Practici	6
Aplicații	7



1. Noțiuni Generale:

Definiție: *Structura* este un tip de dată definită de utilizator, care reunește un grup de variabile între care există o legătură de conținut, sub același nume.

Sintaxă Generală: **struct** ~nume_structură~

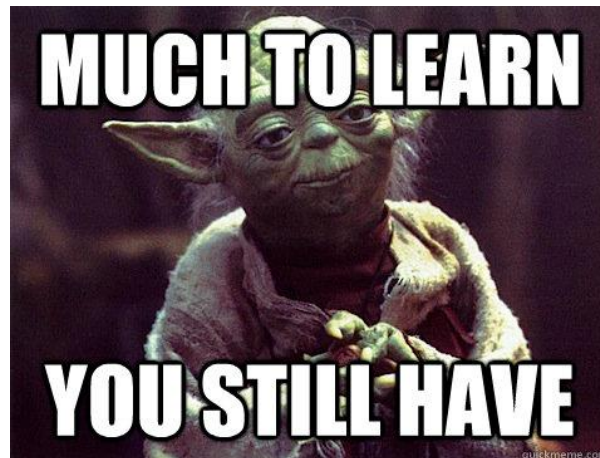


```
{
    <tip_dată1> ~nume_variabilă1~, ~nume_variabilă1~, ...;
    <tip_dată2> ~nume_variabilă2~, ~nume_variabilă2~, ...;
    .....
    <tip_datăn> ~nume_variabilă3~, ~nume_variabilă3~, ...;
};
```

unde “<tip_dată1> ~nume_variabilă1~, ~nume_variabilă1~, ...;” se numește *câmp de înregistrare* sau *membru al înregistrării*. În acest material, vom folosi exclusiv denumirea de “membru”.

De reținut: O structură **poate** avea ca membru o **funcție** (**doar în C++**; în C, acest lucru nu este posibil)! Nu este recomandat să folosiți funcții în cadrul structurilor, deoarece

acest aspect este implementat folosind noțiunea de *clasă* din Programarea Orientată pe Obiecte. Pentru mai multe detalii, puteți să vizitați acest [link](#).



Exemplu de funcție în cadrul unei înregistrări:

```
1 #include <iostream>
2 using namespace std;
3
4 struct dataNastere
5 {
6     int zi, luna, an;
7
8     bool esteAnBisect(int an);
9 };
10
11 bool dataNastere::esteAnBisect(int an)
12 { return an%4 == 0; }
13
14 int main()
15 {
16     dataNastere X = {20, 12, 2020};
17     if(X.esteAnBisect(X.an)) cout << X.an << " este an Bisect!" << endl;
18     else cout << X.an << " NU este an Bisect!" << endl;
19
20     return 0;
21 }
```

```
main.cpp
11
12 struct dataNastere
13 {
14     int zi, luna, an;
15
16     bool esteAnBisect(int an);
17 };
18
19 bool dataNastere::esteAnBisect(int an)
20 { return an%4 == 0; }
21
22 int main()
23 {
24     dataNastere X = {20, 12, 2020};
25     if(X.esteAnBisect(X.an)) cout<<X.an<<" este an Bisect!"<<endl;
26     else cout<<X.an<<" NU este an Bisect!"<<endl;
27
28     return 0;
29 }
30
```

input

2020 este an Bisect!

...Program finished with exit code 0
Press ENTER to exit console.

Un *membru* al unei înregistrări poate fi o *altă înregistrare*. O astfel de înregistrare se numește **structură imbricată**.

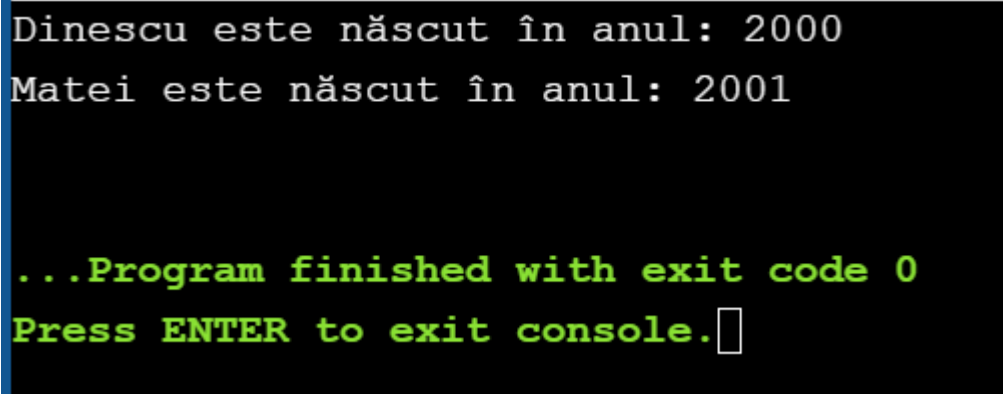


Exemplu:

```

1 #include <iostream>
2 using namespace std;
3
4 // Prima Metodă
5 struct dataNastere
6 {
7     int zi, luna, an;
8 };
9 struct elev1
10 {
11     char nume[30], prenume[50];
12     dataNastere data;
13 };
14
15 // A doua Metodă
16 struct elev2
17 {
18     char nume[30], prenume[50];
19     struct { int zi, luna, an; } dataN;
20 };
21
22 int main()
23 {
24     dataNastere data1 = {30, 03, 2000};
25     elev1 X = {"Dinescu", "Ana", data1};
26     cout << X.nume << " este născut în anul: " << data1.an << endl;
27
28     elev2 Y = {"Matei", "Daniel", {17, 02, 2001}};
29     cout << Y.nume << " este născut în anul: " << Y.dataN.an << endl;
30
31     return 0;
32 }
33

```



```

Dinescu este născut în anul: 2000
Matei este născut în anul: 2001

...Program finished with exit code 0
Press ENTER to exit console.

```

Putem declara și *tablouri de structuri*. Exemplu:

```

1 struct dataNastere
2 {
3     int zi, luna, an;
4 } date[50];

```

2. Bune Practici:

Avem următoarele 2 înregistrări:

```
1 struct aux1
2 {
3     char c;
4     int x;
5     char t;
6     double y;
7     char z;
8 };

1 struct aux2
2 {
3     double y;
4     int x;
5     char c, t, z;
6 };
```

La o primă vedere, aceste 2 structuri par identice (ambele au o singură variabilă *double*, ambele au o singură variabilă *int* și ambele au 3 variabile de tip *char*; singura diferență ar fi poziționarea acestora). Dar oare chiar sunt identice? Hai să rulăm următorul cod:

```
1 int main()
2 {
3     cout << "Aux1 ocupă " << sizeof(aux1) << " de octeți în memorie." << endl;
4     cout << "Aux2 ocupă " << sizeof(aux2) << " de octeți în memorie." << endl;
5
6     return 0;
7 }
```

Compilerul afișează:

```
Aux1 ocupă 32 de octeți în memorie.
Aux2 ocupă 16 de octeți în memorie.

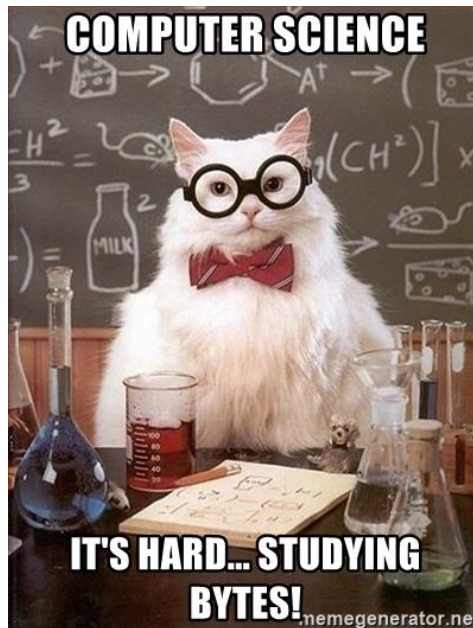
...Program finished with exit code 0
Press ENTER to exit console. □
```

Reamintim faptul că: **char = 1 octet; int = 4 octeți; double = 8 octeți**. Dacă calculăm matematic, ar trebui să obținem

$$3 (3 * \text{char}) + 4 (1 * \text{int}) + 8 (1 * \text{double}) = 15,$$

dar nu obținem acest rezultat în niciunul dintre cazuri. Deci, ce se petrece la nivel intern, în memorie?





Pentru a înțelege, trebuie să cunoaștem cele **3 reguli** de aliniere ale câmpurilor în memorie:

- 1) **Primul câmp declarant se consideră la adresa 0.**
- 2) **Oricare câmp se aliniază la o adresă de memorie care reprezintă un număr egal cu un multiplu al dimensiunii tipului său de dată. Dacă câmpul nu este la o adresă de memorie corespunzătoare, atunci se completează cu octeți aleatori (*padding bits*) până se ajunge la un multiplu.**
- 3) **Dimensiunea (în octeți) a unei structuri trebuie să fie egală cu un multiplu al dimensiunii maxime dintre tipurile de date din interiorul structurii (altfel, se completează cu octeți aleatori).**



Să considerăm cele 2 structuri de mai sus.

```
1 struct aux1
2 {
3     char c;
4     int x;
5     char t;
6     double y;
7     char z;
8 };
```

Reprezentarea în memorie a lui *aux1*:

c	padding bits			x	t	padding bits						y	z	padding bits															
1				4	1							8	1																
								</																					

Luăm variabilele în ordine. 'c' ocupă un octet, de la adresa 0 la adresa 1 (conform [regulei 1](#)). 'x' este variabilă de tip *int*, deci va ocupa 4 octeți. Conform [regulei 2](#), 'x' trebuie să se afle în

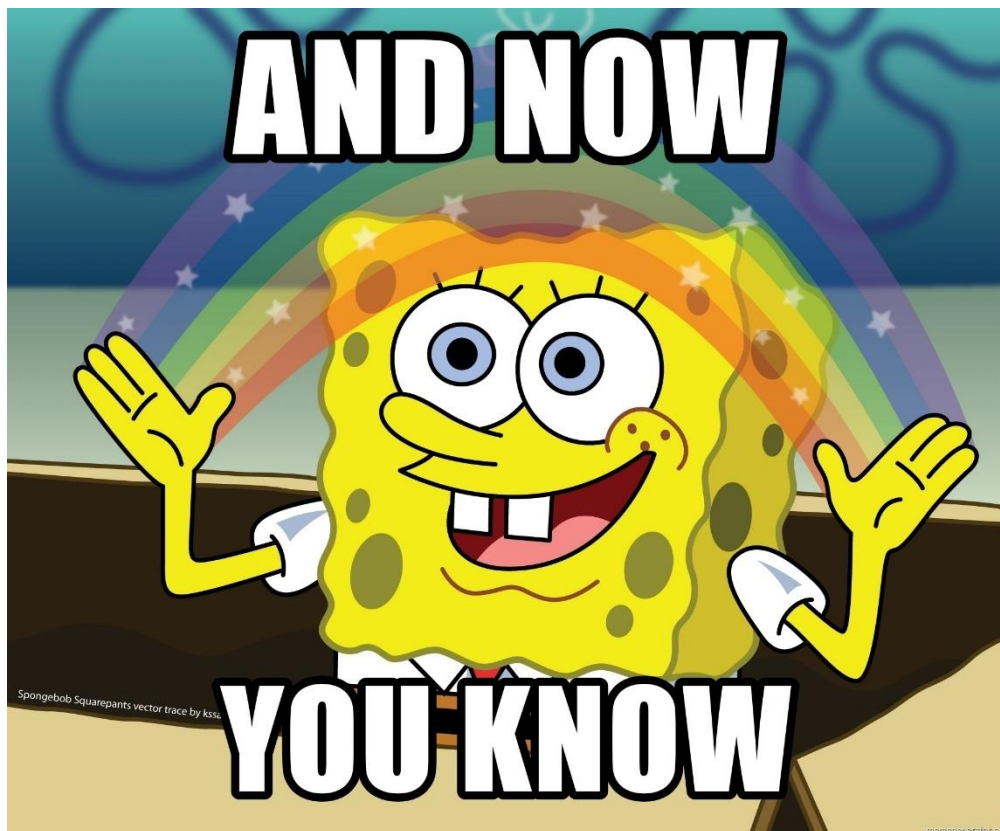
memorie, la o adresă multiplu de 4. Prima adresă multiplu de 4 disponibilă este 4, deci vom ocupa adresele de memorie 1-2, 2-3 și 3-4 cu *biți de padding*, iar de la adresa 4 până la 8 vom avea pe 'x'. Continuăm cu această regulă pentru fiecare variabilă. La final, conformei [regulei 3](#), *aux1* trebuie să ocupe în memorie un număr de octeți egal cu multiplu de 8 (deoarece cea mai mare dimensiune o ocupă *double*, care are 8 biți); deci completăm cu *octeți aleatori* până la adresa **32**.

```
1 struct aux2
2 {
3     double y;
4     int x;
5     char c, t, z;
6 };
```

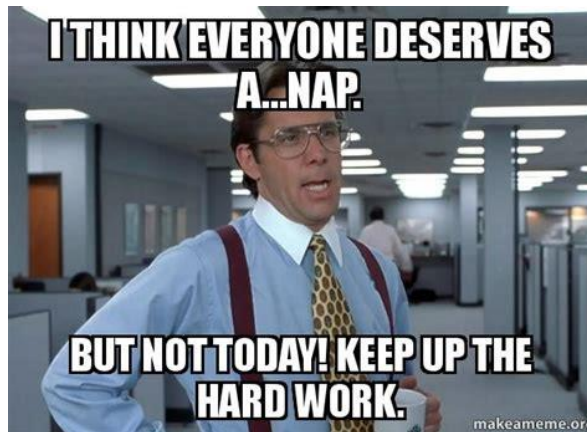
Și acum reprezentăm în memorie pe *aux2*:

y								x				c	t	z	p.b.
8								4				1	1	1	
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

De reținut: Pentru a utiliza cât mai *eficient memoria*, în *structură*, vom declara câmpurile în ordinea descrescătoare a dimensiunii tipurilor de date, așa cum este prezentat *aux2*!



3. Aplicații:



3.1. Se citesc de la tastatură 2 numere complexe. Să se implementeze operațiile de adunare, scădere, înmulțire și împărțire.

Aspecte Preliminarii: Detalii despre lucrul cu numerele complexe se găsesc la acest [link](#).
Dacă $x = a_1 + b_1i$ și $y = a_2 + b_2i$ atunci:

- Formulă *Înmulțire*: $xy = (a_1a_2 - b_1b_2) + (a_1b_2 + b_1a_2)i$
- Formulă *Împărțire*: $x / y = \frac{(a_1a_2 + b_1b_2) + (b_1a_2 - a_1b_2)i}{a_2^2 + b_2^2}$

Reprezentarea Datelor Problemei:

Putem reprezenta un număr complex astfel:

```
1 struct nrComplex
2 {
3     double r, i; // r -> partea reală; i -> cea imaginară
4 };
```

Exemplu Date de Intrare: $x = 2.5 - 7i$; $y = 7.8 + 2.3i$

Exemplu Date de Iesire: $x + y = 10.3 - 4.7i$; $x - y = -5.3 - 9.3i$; $xy = 35.6 - 48.85i$;

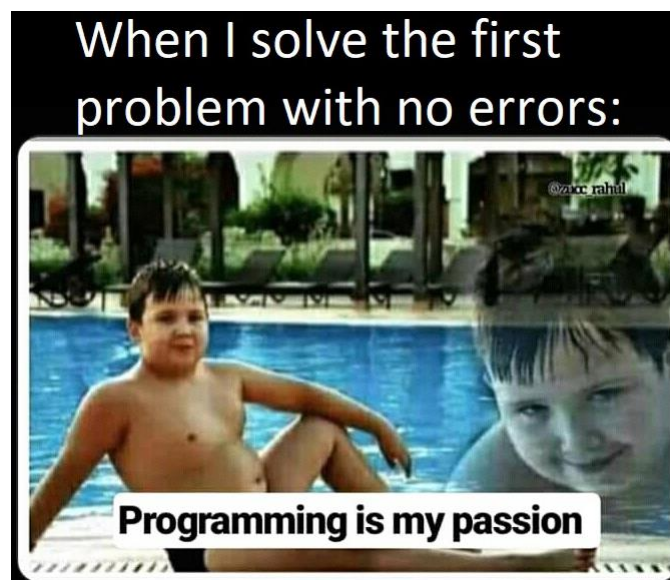
$$x / y = \frac{3.4 - 60.35i}{66.13}$$

Rezolvare:

```
1 #include <iostream>
2 #include <math.h>
3 using namespace std;
4
5 struct nrComplex
6 {
7     double r, i; // r -> partea reală; i -> cea imaginară
8 };
9
10 int main()
11 {
12     nrComplex x, y;
13     cout << "x: "; cin >> x.r >> x.i;
14     cout << "y: "; cin >> y.r >> y.i;
```



```
15
16     double aux1, aux2, aux3;
17     aux1 = x.r + y.r;
18     aux2 = x.i + y.i;
19     cout << "x + y = " << aux1;
20     if (aux2 >= 0) cout << '+' << aux2 << 'i' << endl;
21     else cout << aux2 << 'i' << endl;
22
23     aux1 = x.r - y.r;
24     aux2 = x.i - y.i;
25     cout << "x - y = " << aux1;
26     if (aux2 >= 0) cout << '+' << aux2 << 'i' << endl;
27     else cout << aux2 << 'i' << endl;
28
29     aux1 = (x.r * y.r) - (x.i * y.i);
30     aux2 = (x.r * y.i) + (y.r * x.i);
31     cout << "x * y = " << aux1;
32     if (aux2 >= 0) cout << '+' << aux2 << 'i' << endl;
33     else cout << aux2 << 'i' << endl;
34
35     aux1 = (x.r * y.r) + (x.i * y.i);
36     aux2 = (x.i * y.r - x.r * y.i);
37     aux3 = pow(y.r, 2) + pow(y.i, 2);
38     cout << "x / y = " << '(' << aux1;
39     if (aux2 >= 0) cout << '+' << aux2 << "i) / " << aux3 << endl;
40     else cout << aux2 << "i) / " << aux3 << endl;
41
42     return 0;
43 }
```



3.2. Fie $n \in \mathbb{N}$ și fie n triplete (A_i, B_i, C_i) de puncte în plan cu $i = 1, \dots, n$. Fiecare punct are coordonatele (X, Y) , numere reale. Să se afișeze cea mai mare arie a unui triunghi, dacă aceasta există, sau mesajul “Nu există!” dacă niciun triplet de puncte nu formează un triunghi.

Aspecte Preliminarii:

- Formulă de calcul pentru *lungimea unui segment*, cunoscându-se două puncte:

$$AB = \sqrt{(X_A - X_B)^2 + (Y_A - Y_B)^2}$$

- *Aria unui triunghi*, cunoscând doar lungimea segmentelor ([Formula lui Heron](#)):

$$A_{ABC} = \sqrt{p(p-a)(p-b)(p-c)}; \text{ unde } p = \frac{(a+b+c)}{2} \text{ (semiperimetrul)}$$

- Un *triunghi este valid* dacă *vârfurile acestuia nu sunt coliniare sau identice*, ceea ce este echivalent cu a spune că un *triunghi este valid* dacă *suma oricăror două laturi este diferită de a treia*.

Reprezentarea Datelor Problemei:

Putem folosi 2 structuri pentru a reprezenta punctele și triunghiurile. Astfel:

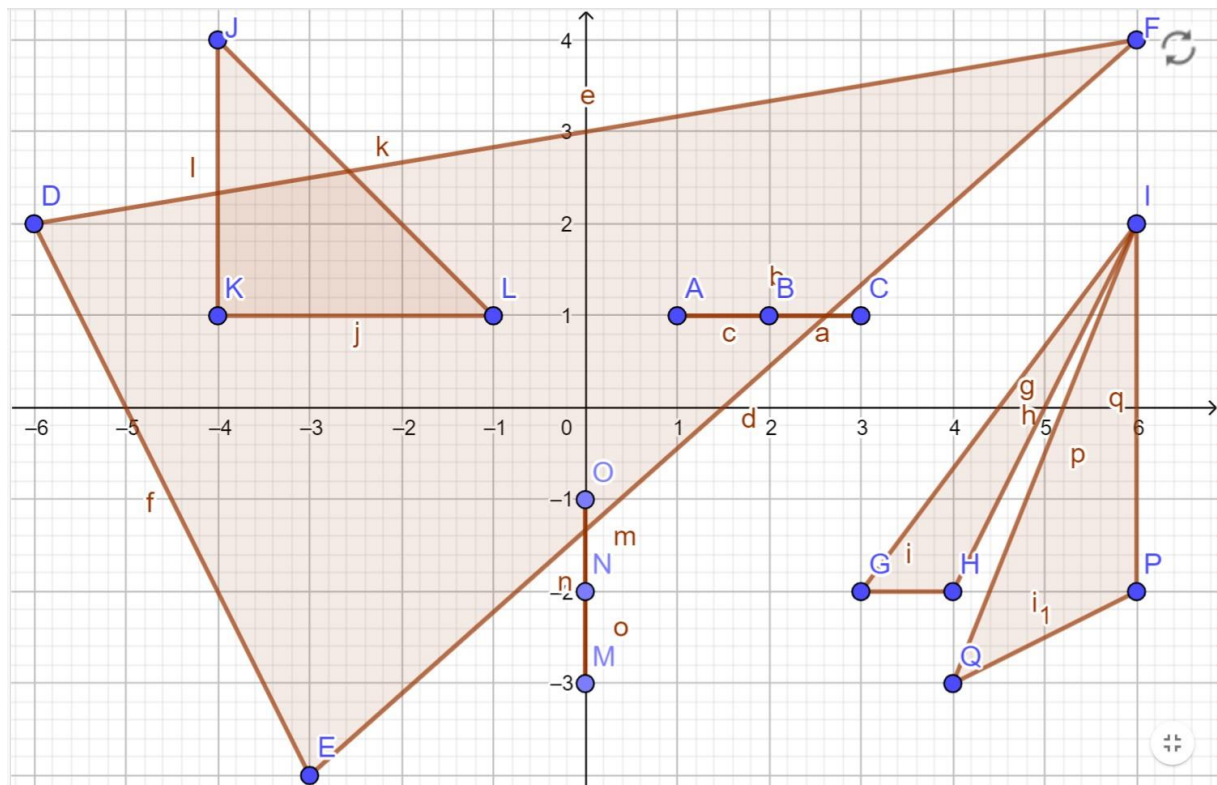
- Pentru punct:

```
1 struct Punct
2 { double X, Y; };
```

- Pentru triunghi:

```
1 struct Triunghi
2 {
3     Punct A, B, C;
4     double AB, BC, AC;
5     double p; //semiperimetru
6     double arie; //arie
7     bool esteValid;
8 };
```

Exemplu Date de Intrare:



Este evident că aria maximă este cea a triunghiului DEF.

Rezolvare 1: Parcurgem tabloul de structuri și reținem aria maximă.

```

1 #include <iostream>
2 #include <math.h>
3 using namespace std;
4
5 struct Punct
6 { double X, Y; };
7 struct Triunghi
8 {
9     Punct A, B, C;
10    double AB, BC, AC;
11    double p; //semiperimetru
12    double arie; //arie
13    bool esteValid;
14 } t[10];
15
16 bool validare(Triunghi &t)
17 {
18     t.AB = sqrt(pow(t.A.X - t.B.X, 2) + pow(t.A.Y - t.B.Y, 2));
19     t.AC = sqrt(pow(t.A.X - t.C.X, 2) + pow(t.A.Y - t.C.Y, 2));
20     t.BC = sqrt(pow(t.B.X - t.C.X, 2) + pow(t.B.Y - t.C.Y, 2));
21
22     // Verificăm dacă este Triunghi
23     if(t.AB + t.BC == t.AC || t.AB + t.AC == t.BC
24        || t.BC + t.AC == t.AB)
25         return false;

```

```

26
27     return true;
28 }
29
30 void calcul(Triunghi &t)
31 {
32     t.p = (t.AB + t.BC + t.AC) / 2;
33     t.arie = sqrt(t.p * (t.p - t.AB) * (t.p - t.BC) * (t.p - t.AC));
34 }
35
36
37
38 int main()
39 {
40     // Date de Intrare
41     t[0] = {{-6, 2}, {-3, -4}, {6, 4}};
42     t[1] = {{-4, 4}, {-4, 1}, {-1, 1}};
43     t[2] = {{-1, 0}, {-2, 0}, {-3, 0}}; // puncte coliniare
44     t[3] = {{1, 1}, {1, 1}, {3, 1}}; // puncte identice
45     t[4] = {{6, 1}, {3, -2}, {4, -2}};
46     t[5] = {{6, 1}, {4, -3}, {6, -2}};
47
48     int i;
49     double maxim = 0;
50     for(i = 0; i < 6; i++)
51         t[i].esteValid = validare(t[i]);
52
53     cout << "Ariile Triunghiurilor: " << endl;
54     for(i = 0; i < 6; i++)
55         if(t[i].esteValid)
56         {
57             calcul(t[i]);
58             cout << i + 1 << ": " << t[i].arie << endl;
59             if (maxim < t[i].arie) maxim = t[i].arie;
60         }
61
62     if(maxim) cout << endl << "Aria Maximă este: " << maxim << endl;
63     else cout << endl << "Nu există!" << endl;
64
65     return 0;
66 }

```

```

Ariile Triunghiurilor:
1: 39
2: 4.5
5: 1.5
6: 3

Aria Maximă este: 39

```

Rezolvare 2: Sortăm descrescător tabloul de structuri și aria maximă se va afla pe prima poziție. Pentru informații despre sort, accesați acest [link](#).

```

1 #include <iostream>
2 #include <math.h>
3 #include <algorithm> //pentru sort
4 using namespace std;
5
6 struct Punct
7 { double X, Y; };
8 struct Triunghi
9 {
10     Punct A, B, C;
11     double AB, BC, AC;
12     double p; //semiperimetru
13     double arie; //arie
14     bool esteValid;
15 } t[10];
16
17 bool validare(Triunghi &t)
18 {
19     t.AB = sqrt(pow(t.A.X - t.B.X, 2) + pow(t.A.Y - t.B.Y, 2));
20     t.AC = sqrt(pow(t.A.X - t.C.X, 2) + pow(t.A.Y - t.C.Y, 2));
21     t.BC = sqrt(pow(t.B.X - t.C.X, 2) + pow(t.B.Y - t.C.Y, 2));
22
23     // Verificăm dacă este Triunghi
24     if(t.AB + t.BC == t.AC || t.AB + t.AC == t.BC
25        || t.BC + t.AC == t.AB)
26         return false;
27
28     return true;
29 }
30
31 void calcul(Triunghi &t)
32 {
33     t.p = (t.AB + t.BC + t.AC) / 2;
34     t.arie = sqrt(t.p * (t.p - t.AB) * (t.p - t.BC) * (t.p - t.AC));
35 }
36
37 // Funcție de comparare a triunghiurilor folosită la sortare
38 bool compara(Triunghi a, Triunghi b)
39 { return a.arie > b.arie; }
40
41 int main()
42 {
43     // Date de Intrare
44     t[0] = {{-6, 2}, {-3, -4}, {6, 4}};
45     t[1] = {{-4, 4}, {-4, 1}, {-1, 1}};
46     t[2] = {{-1, 0}, {-2, 0}, {-3, 0}}; // puncte coliniare
47     t[3] = {{1, 1}, {1, 1}, {3, 1}};    // puncte identice
48     t[4] = {{6, 1}, {3, -2}, {4, -2}};
49     t[5] = {{6, 1}, {4, -3}, {6, -2}};
50
51     int i;
52     for(i = 0; i < 6; i++)
53         t[i].esteValid = validare(t[i]);
54
55     for(i = 0; i < 6; i++)
56         if(t[i].esteValid)

```

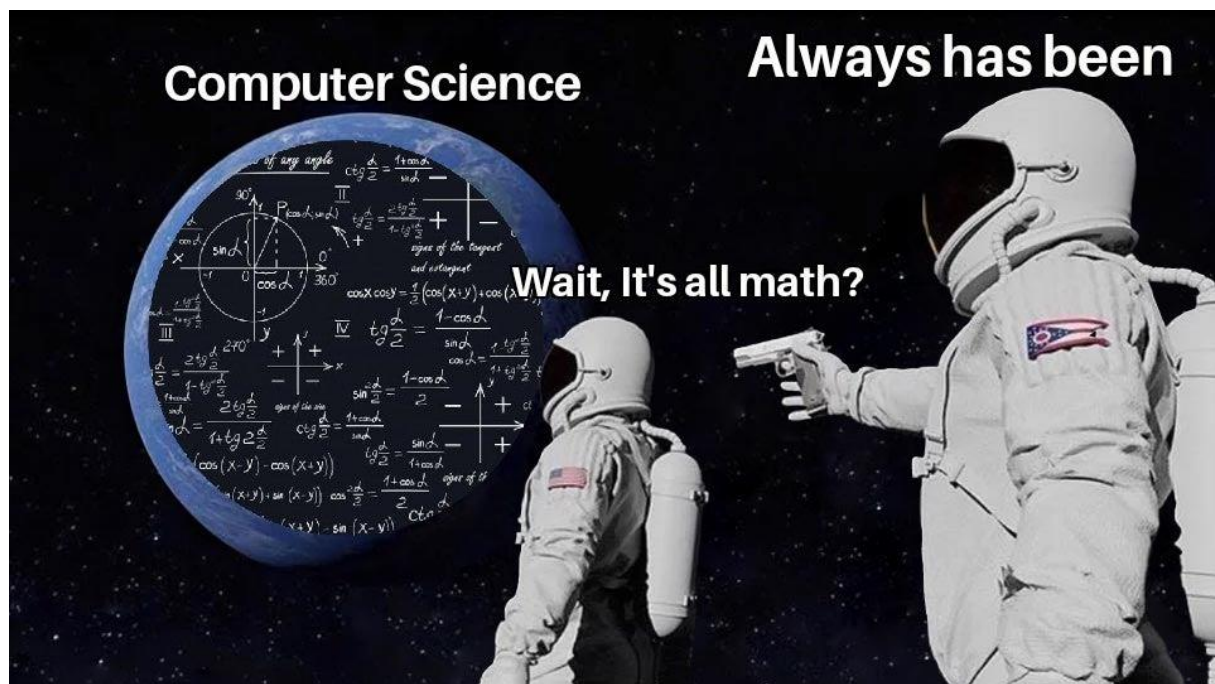
```

57         calcul(t[i]);
58
59         // sortăm vectorul descrescător -> avem aria maximă pe prima poziție
60         sort(&t[0], &t[6], compara);
61         cout << "Vectorul Sortat: ";
62         for(i = 0; i < 6; i++) cout << t[i].arie << " ";
63         cout << endl;
64
65         if(t[0].arie)
66             cout << endl << "Aria Maximă este: " << t[0].arie << endl;
67         else cout << endl << "Nu există!" << endl;
68
69         return 0;
70 }

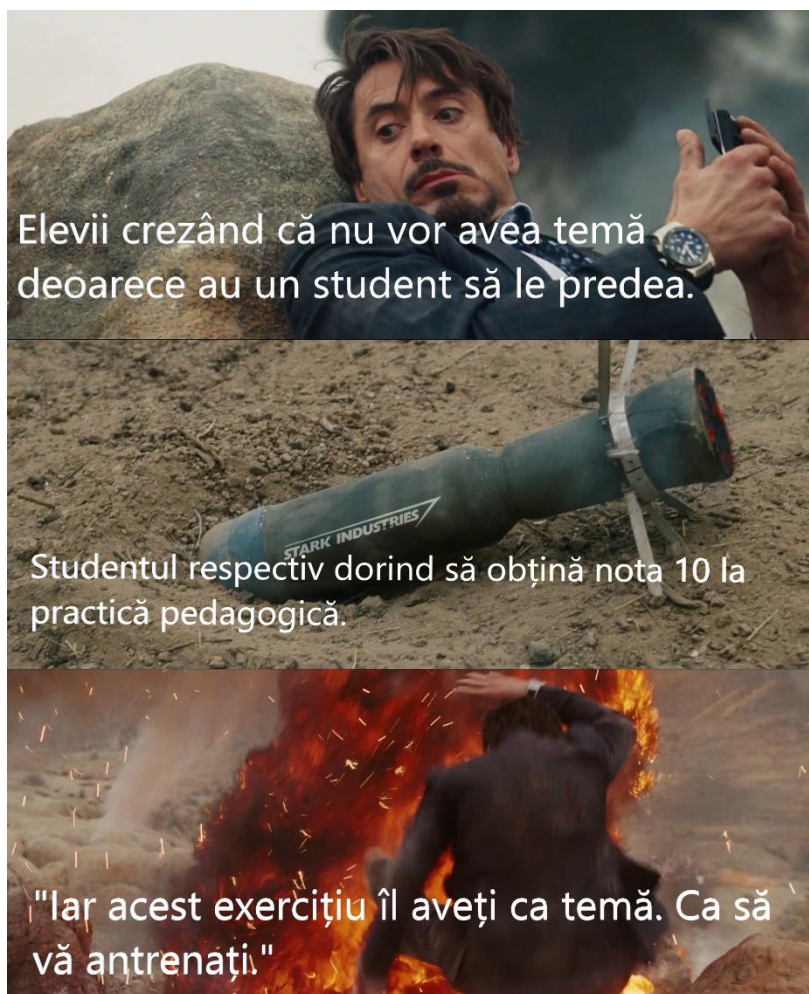
```

Vectorul Sortat: 39 4.5 3 1.5 0 0

Aria Maximă este: 39



3.3. Temă:



La un concurs au participat n persoane, fiecare câștigând o anumită sumă de bani. Scrieți un program care citește dintr-un fișier numărul n de participanți și cei n participanți caracterizați prin nume, oraș și suma câștigată și care afișează:

- suma premiilor din fiecare oraș;
- participanții ordonați alfabetic după nume.

Exemplu:

Date de Intrare	Date de Ieșire
6 Ionescu București 100 Florescu Arad 70 Iocai Cluj 90 Antal Arad 120 Serghei București 95 Matei Arad 85	a) București 195 Arad 275 Cluj 90 b) Antal Arad 120 Florescu Arad 70 Iocai Cluj 90 Ionescu București 100 Matei Arad 85 Serghei București 95

