

1. Gândirea computațională

Definiție. Gândirea computațională este modul de abordare a unei probleme prin care aceasta este pregătită pentru a fi rezolvată cu ajutorul calculatorului.

Gândirea computațională are la bază operațiile prin care un enunț scris în cuvinte devine o succesiune de prelucrări:

- **descompunerea** — problema este împărțită în subprobleme mai simple, fiecare rezolvată separat;
- **recunoașterea tiparelor** — sunt identificate prelucrările care se repetă sau care seamănă cu prelucrări cunoscute;
- **abstractizarea** — sunt reținute numai datele și relațiile care intervin în rezolvare, iar restul sunt lăsate deoparte;
- **algoritmizarea** — este stabilită succesiunea de pași prin care se ajunge de la datele de intrare la rezultat.

1.1. Cele patru operații pe un exemplu

Exemplu. Într-un registru de casă sunt trecute n operații. Fiecare operație este un număr întreg: pozitiv pentru o încasare, negativ pentru o cheltuială. Să se determine totalul încasărilor, totalul cheltuielilor și soldul.

- **Descompunerea:** problema cere trei rezultate, iar fiecare poate fi obținut separat.
- **Recunoașterea tiparelor:** totalul încasărilor și totalul cheltuielilor sunt obținute prin aceeași prelucrare, adunarea repetată, aplicată unor valori diferite.
- **Abstractizarea:** data operației, persoana care a făcut plata și motivul ei nu intervin în calcul, deci nu sunt reținute.
- **Algoritmizarea:** sumele sunt parcurse una câte una, iar fiecare este adăugată la totalul potrivit.

2. Analiza problemei

Analiza răspunde la două întrebări: ce se cunoaște și ce se cere. Valorile care apar în rezolvare sunt împărțite după rolul lor.

Felul datelor	Ce sunt	Unde apar în program
Date de intrare	Valorile cunoscute, primite de program din exterior.	Sunt citite.
Date de ieșire	Rezultatele cerute de enunț.	Sunt afișate.
Date de manevră	Valorile folosite numai în timpul prelucrării, pentru obținerea rezultatelor.	Nu sunt nici citite, nici afișate.

2.1. Datele de manevră

Datele de manevră sunt cele care se recunosc cel mai greu, pentru că nu apar în enunț. La interschimbarea valorilor a două variabile, variabila `aux` este dată de manevră:

PYTHON

```
aux = a
a = b
b = aux
```

C++

```
aux = a;
a = b;
b = aux;
```

Valoarea ei nu este citită și nu este afișată. Fără ea, prima atribuire ar pierde valoarea variabilei `a`. Tot date de manevră sunt contorul unei structuri repetitive și variabilele în care se adună rezultatele parțiale.

2.2. Restricțiile

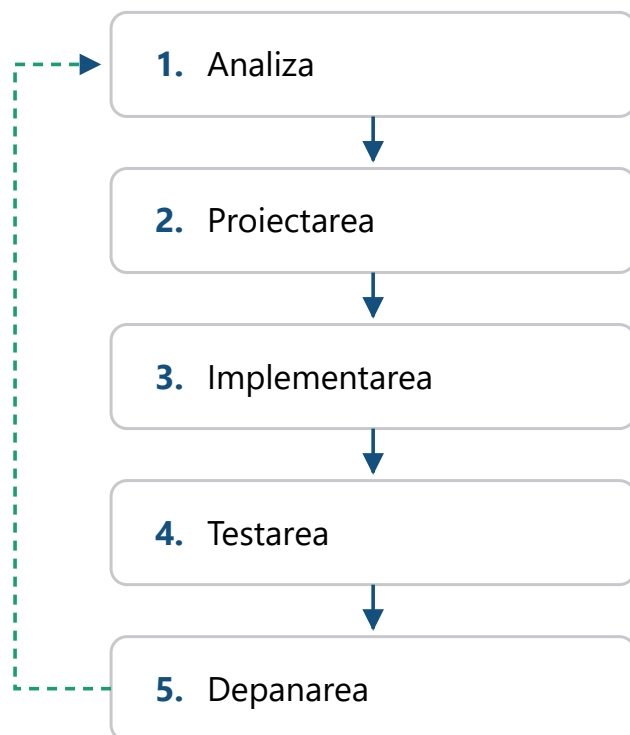
Enunțul stabilește și limitele între care se află datele: câte valori sunt citite, ce semn au, între ce margini se încadrează. Din restricții se obțin cazurile-limită pe care programul este testat și tipurile de date folosite la declarare.

Analiza problemei registrului de casă.

- **Date de intrare:** n — numărul de operații; cele n sume, numere întregi.
- **Date de ieșire:** totalul încasărilor, totalul cheltuielilor și soldul.
- **Date de manevră:** i — contorul structurii repetitive; suma — valoarea citită la pasul curent.
- **Restricții:** n este un număr natural nenul; o sumă egală cu 0 nu este nici încasare, nici cheltuială.

3. Etapele elaborării unui program

Elaborarea unui program parcurge cinci etape. Fiecare pornește de la rezultatul celei dinainte, iar erorile descoperite la testare duc la reluarea etapelor anterioare.



Erorile descoperite la testare trimit înapoi la analiză sau la proiectare, nu numai la programul sursă.

Etapa	În ce constă	Ce rezultă
1. Analiza	Sunt stabilite datele de intrare, datele de ieșire și restricțiile impuse de enunț.	Enunțul, scris în termeni de date
2. Proiectarea	Problema este descompusă în module, iar pentru fiecare este aleasă metoda de rezolvare.	Algoritmul
3. Implementarea	Algoritmul este transcris în limbajul de programare, respectând regulile lui de scriere.	Programul sursă
4. Testarea	Programul este executat pe seturi de date pentru care rezultatul este cunoscut dinainte.	Erorile descoperite
5. Depanarea	Sunt căutate cauzele erorilor descoperite la testare și sunt înlăturate.	Programul corectat

Testarea arată *că* programul dă un rezultat greșit. Depanarea arată *de ce*. Cele două sunt etape diferite: o eroare găsită la testare poate avea cauza în algoritm, deci în etapa de proiectare, nu în codul scris.

4. Etapele parcurse pe o problemă

Cele cinci etape sunt parcurse mai jos pe problema registrului de casă, enunțată în prima secțiune.

4.1. Analiza

Datele au fost stabilite în secțiunea a doua: se citesc n și cele n sume, se afișează totalul încasărilor, totalul cheltuielilor și soldul.

4.2. Proiectarea

Sumele sunt parcurse o singură dată, iar semnul fiecăreia arată la care total este adăugată. Algoritmul, scris în pseudocod:

```
citește n
incasari ← 0
cheltuieli ← 0
pentru i ← 1,n execută
    citește suma
    dacă suma > 0 atunci
        incasari ← incasari + suma
    altfel
        cheltuieli ← cheltuieli - suma
sfdacă
sfpentru
sold ← incasari - cheltuieli
scrie incasari, cheltuieli, sold
```

Cheltuielile sunt adunate cu semn schimbat, `cheltuieli - suma`, pentru că sumele negative reprezintă plăți. Totalul cheltuielilor este astfel un număr pozitiv.

4.3. Implementarea

Algoritmul este transcris în cele două limbaje:

PYTHON

```
n = int(input())

incasari = 0
cheltuieli = 0

for i in range(n):
    suma = int(input())
    if suma > 0:
        incasari = incasari + suma
    else:
        cheltuieli = cheltuieli -
suma

sold = incasari - cheltuieli

print(incasari)
print(cheltuieli)
print(sold)
```

C++

```
#include <iostream>
using namespace std;

int main() {
    int n, suma;
    int incasari = 0, cheltuieli =
0;
    cin >> n;
    for (int i = 1; i <= n; i++) {
        cin >> suma;
        if (suma > 0)
            incasari = incasari +
suma;
        else
            cheltuieli = cheltuieli
- suma;
    }
    cout << incasari << '\n';
    cout << cheltuieli << '\n';
    cout << incasari - cheltuieli <<
'\n';
    return 0;
}
```

Algoritmul este același, iar deosebirile țin de regulile de scriere ale fiecărui limbaj: declararea variabilelor și tipul lor, marcarea blocurilor prin acolade sau prin indentare, forma structurii repetitive cu contor.

4.4. Testarea

Programul este executat pe seturi de date pentru care rezultatele sunt cunoscute dinainte. Între ele se află și cazurile-limită obținute din restricții.

Date de intrare	Rezultat așteptat	Ce verifică
3 120 -50 -30	120, 80, 40	caz obișnuit
2 40 60	100, 0, 100	numai încasări
2 -25 -75	0, 100, -100	numai cheltuieli
1 0	0, 0, 0	valoarea 0, caz-limită

4.5. Depanarea

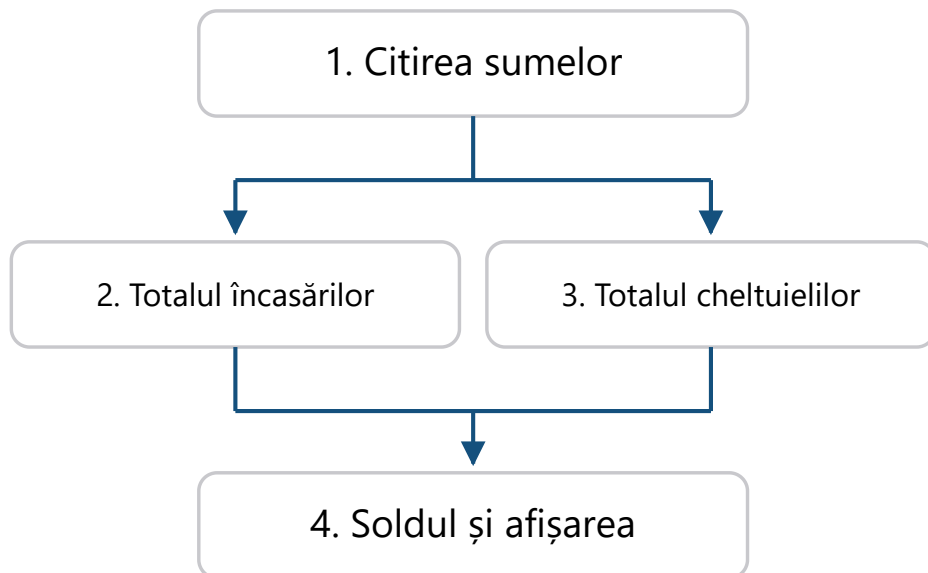
Depanarea pornește de la un test căzut și caută instrucțiunea care produce valoarea greșită. Valorile variabilelor sunt urmărite pas cu pas, fie prin afișări puse temporar în program, fie cu depanatorul mediului de programare. Felurile de erori și semnele după care sunt recunoscute se află în secțiunea a șasea.

5. Proiectarea modulară

Definiție. Un modul este o parte de sine stătătoare a rezolvării, care îndeplinește o singură sarcină și care comunică cu restul programului prin datele primite și prin rezultatul întors.

Proiectarea modulară înseamnă descompunerea problemei în module înainte de scrierea codului. Pentru problema registrului de casă, descompunerea poate fi:

Modulul	Ce face
1. Citirea sumelor	Preia de la intrare n și cele n valori.
2. Totalul încasărilor	Adună sumele pozitive.
3. Totalul cheltuielilor	Adună sumele negative, luate cu semn schimbat.
4. Soldul și afișarea	Scade cele două totaluri și afișează rezultatele.



Săgețile arată datele care trec de la un modul la altul: sumele citite, apoi cele două totaluri.

5.1. Două organizări ale aceleiași rezolvări

Aceeași problemă poate fi descompusă în mai multe feluri. Organizarea **A** calculează fiecare total într-un modul propriu, care parcurge sumele memorate; organizarea **B** parcurge sumele o singură dată și actualizează amândouă totalurile la fiecare pas.

Criteriul	Organizarea A — un modul pentru fiecare total	Organizarea B — o singură parcurgere
Parcurgerile datelor	Cele n sume sunt parcurse de două ori, o dată pentru fiecare total.	Cele n sume sunt parcurse o singură dată.
Memoria necesară	Cele n sume sunt memorate, ca să poată fi parcurse din nou.	Se memorează numai cele două totaluri, indiferent cât de mare este n .
Claritatea	Fiecare total are modulul lui, iar rolul fiecărui modul se citește din numele său.	Un singur modul face două prelucrări deodată, deci rolul lui se citește mai greu.
Adăugarea unui rezultat nou	Se scrie un modul nou, fără a fi modificate cele existente.	Se modifică modulul care parcurge sumele.

Niciuna dintre organizări nu este mai bună în toate privințele. Organizarea B este mai eficientă, iar programul din secțiunea a patra o urmează. Organizarea A se citește mai ușor și se modifică mai ușor, iar pentru date de dimensiuni mici deosebirea de timp nu se observă. Criteriul după care se alege este cerința problemei: volumul datelor, numărul rezultatelor cerute și felul în care aplicația va fi extinsă.

6. Încadrarea erorilor

Erorile dintr-un program se încadrează în trei categorii, după momentul în care apar și după felul în care se manifestă.

Categoria	Când apare	Cum se manifestă
Eroare de sintaxă	Înainte de executare.	Textul programului încalcă regulile de scriere ale limbajului. Programul nu pornește.
Eroare de logică	În timpul executării, fără mesaj.	Programul se execută până la capăt, dar rezultatul este greșit. Algoritmul nu rezolvă problema cerută.
Eroare de executare	În timpul executării, cu oprire.	Programul întâlnește o operație care nu poate fi efectuată și se oprește înainte de final.

6.1. Eroare de sintaxă

Lipsește semnul cerut de regulile de scriere ale limbajului:

PYTHON

```
n = int(input())
if n > 0
    print(n)
```

C++

```
int n;
cin >> n
cout << n;
```

În Python lipsește caracterul `:` de la sfârșitul condiției, iar în C++ lipsește caracterul `;` de la sfârșitul instrucțiunii. Programul nu pornește: eroarea este semnalată înainte de executare.

6.2. Eroare de logică

Programul următor ar trebui să afișeze media a n numere întregi, dar afișează câtul împărțirii întregi:

PYTHON

```
n = int(input())
suma = 0
for i in range(n):
    suma = suma + int(input())

print(suma // n)
```

C++

```
int n, suma = 0, x;
cin >> n;
for (int i = 1; i <= n; i++) {
    cin >> x;
    suma = suma + x;
}
cout << suma / n;
```

Pentru valorile 8, 9, 7 și 10 se obține 8 în loc de 8.5. Programul se execută până la capăt și nu afișează niciun mesaj de eroare, deci greșeala se descoperă numai comparând rezultatul cu cel așteptat. În Python, operatorul `//` dă câtul împărțirii întregi, iar `/` dă rezultat real. În C++, operatorul `/` aplicat la doi operanzi întregi dă tot câtul împărțirii întregi, deci unul dintre operanzi trebuie convertit la un tip real.

6.3. Eroare de executare

Chiar corectat, programul de mai sus se oprește atunci când n are valoarea 0, pentru că împărțirea la zero nu poate fi efectuată:

```
n = int(input())
suma = 0
for i in range(n):
    suma = suma + int(input())

print(suma / n)
```

În Python programul se oprește cu mesajul `ZeroDivisionError`. În C++, împărțirea întreagă la zero nu are un rezultat definit de limbaj, iar programul se oprește fără ca eroarea să fi fost semnalată la compilare. Eroarea se înlătură verificând valoarea lui n înainte de împărțire, așa cum se face la problema temperaturilor din activitatea practică.

7. Activitate practică

Pentru fiecare problemă se stabilesc mai întâi datele de intrare, de ieșire și de manevră, apoi se scrie algoritmul în pseudocod și abia la urmă programul, în Python și în C++.

1. Media notelor

Se citește n , apoi n note. Să se afișeze media lor, rotunjită la două zecimale.

Intrare: 4 | 8 9 7 10 → Ieșire: media 8,5

Intrare: n și cele n note. Ieșire: media. Manevră: contorul i și variabila suma, în care se adună notele. Împărțirea trebuie să dea rezultat real, deci variabila suma este declarată de tip real. Programul Python afișează 8.5, iar cel în C++ afișează 8.50, pentru că numărul zecimalelor este fixat prin `setprecision`.

PYTHON

```
n = int(input())

suma = 0
for i in range(n):
    nota = float(input())
    suma = suma + nota

print(round(suma / n, 2))
```

C++

```
#include <iostream>
#include <iomanip>
using namespace std;

int main() {
    int n;
    double nota, suma = 0;
    cin >> n;
    for (int i = 1; i <= n; i++) {
        cin >> nota;
        suma = suma + nota;
    }
    cout << fixed <<
    setprecision(2) << suma / n;
    return 0;
}
```

2. Maximul și numărul aparițiilor

Se citește n , apoi n numere întregi. Să se afișeze cea mai mare valoare și de câte ori apare ea.

Intrare: 5 | 3 9 2 9 1 → Ieșire: 9 2

Intrare: n și cele n numere. Ieșire: maximul și numărul aparițiilor lui. Manevră: contorul i , valoarea citită x , variabilele maxim și aparitii . Maximul se inițializează cu prima valoare citită, nu cu zero: numerele pot fi negative.

PYTHON

```
n = int(input())

maxim = int(input())
aparitii = 1

for i in range(n - 1):
    x = int(input())
    if x > maxim:
        maxim = x
        aparitii = 1
    elif x == maxim:
        aparitii = aparitii + 1

print(maxim, aparitii)
```

C++

```
#include <iostream>
using namespace std;

int main() {
    int n, x, maxim, aparitii = 1;
    cin >> n >> maxim;
    for (int i = 1; i <= n - 1; i++) {
        cin >> x;
        if (x > maxim) {
            maxim = x;
            aparitii = 1;
        } else if (x == maxim)
            aparitii = aparitii + 1;
    }
    cout << maxim << ' ' << aparitii;
    return 0;
}
```

3. Temperaturile unei săptămâni

Se citește n , apoi temperaturile înregistrate în n zile. Să se afișeze câte zile au avut temperatura negativă și media temperaturilor pozitive. Dacă nu a existat nicio temperatură pozitivă, se afișează mesajul „fara temperaturi pozitive”.

Intrare: 5 | -3 4 0 6 -1 → Ieșire: 2 zile și media 5

Intrare: n și cele n temperaturi. Ieșire: numărul zilelor cu temperatură negativă și media temperaturilor pozitive. Manevră: contorul i , temperatura citită t , variabilele suma_pozitive și numar_pozitive . Cazul-limită apare când nicio temperatură nu este pozitivă: împărțirea ar fi făcută la zero.

PYTHON

```
n = int(input())

zile_negative = 0
suma_pozitive = 0
numar_pozitive = 0

for i in range(n):
    t = float(input())
    if t < 0:
        zile_negative =
zile_negative + 1
    elif t > 0:
        suma_pozitive =
suma_pozitive + t
        numar_pozitive =
numar_pozitive + 1

print(zile_negative)
if numar_pozitive > 0:
    print(round(suma_pozitive /
numar_pozitive, 2))
else:
    print("fara temperaturi
pozitive")
```

C++

```
#include <iostream>
#include <iomanip>
using namespace std;

int main() {
    int n, zile_negative = 0,
numar_pozitive = 0;
    double t, suma_pozitive = 0;
    cin >> n;
    for (int i = 1; i <= n; i++) {
        cin >> t;
        if (t < 0)
            zile_negative =
zile_negative + 1;
        else if (t > 0) {
            suma_pozitive =
suma_pozitive + t;
            numar_pozitive =
numar_pozitive + 1;
        }
    }
    cout << zile_negative << '\n';
    if (numar_pozitive > 0)
        cout << fixed <<
setprecision(2) << suma_pozitive /
numar_pozitive;
    else
        cout << "fara temperaturi
pozitive";
    return 0;
}
```

4. Cea mai lungă serie de încasări

Se citește n , apoi n operații de casă, ca la problema din lecție. Să se afișeze lungimea celei mai lungi serii de încasări consecutive.

Intrare: 7 | 10 20 -5 30 40 50 -1 → Ieșire: 3

Intrare: n și cele n sume. Ieșire: lungimea celei mai lungi serii de valori pozitive consecutive.

Manevră: contorul i , suma citită, seria curentă și seria cea mai lungă. Seria curentă se reia de la zero la fiecare valoare care nu este încasare, iar seria cea mai lungă se actualizează la fiecare pas.

PYTHON

```
n = int(input())

serie = 0
maxim = 0

for i in range(n):
    suma = int(input())
    if suma > 0:
        serie = serie + 1
        if serie > maxim:
            maxim = serie
    else:
        serie = 0

print(maxim)
```

C++

```
#include <iostream>
using namespace std;

int main() {
    int n, suma, serie = 0, maxim = 0;
    cin >> n;
    for (int i = 1; i <= n; i++) {
        cin >> suma;
        if (suma > 0) {
            serie = serie + 1;
            if (serie > maxim)
                maxim = serie;
        } else
            serie = 0;
    }
    cout << maxim;
    return 0;
}
```

8. Ce trebuie să reținem

- Gândirea computațională are la bază descompunerea problemei, recunoașterea tiparelor, abstractizarea și algoritmizarea.
- Analiza stabilește datele de intrare, datele de ieșire și restricțiile din enunț.
- Datele de manevră nu sunt nici citite, nici afișate: ele există numai în timpul executării programului.

- Elaborarea unui program parcurge cinci etape: analiza, proiectarea, implementarea, testarea și depanarea.
- Rezultatul proiectării este algoritmul, iar rezultatul implementării este programul sursă.
- Un modul îndeplinește o singură sarcină, iar legătura lui cu celelalte module se face prin datele primite și prin rezultatul întors.
- Aceeași problemă poate fi descompusă în mai multe feluri, iar organizarea aleasă schimbă numărul parcurgerilor, memoria necesară și claritatea rezolvării.
- Erorile de sintaxă opresc programul înainte de executare, cele de logică dau un rezultat greșit fără mesaj, iar cele de executare opresc programul în timpul lui.
- Testarea se face pe seturi de date pentru care rezultatul este cunoscut dinainte, inclusiv pe cazuri-limită.