

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.

Termenul **modul** numește o parte de sine stătătoare a rezolvării, care îndeplinește o singură sarcină. Descompunerea unei probleme în module este pasul prin care o rezolvare lungă devine un șir de rezolvări scurte.

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:

```
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

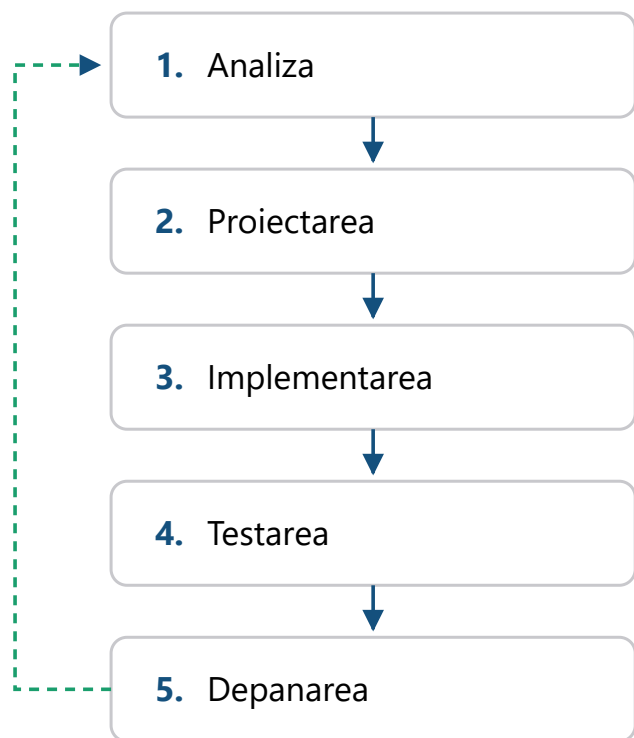
Enunțul stabilește și limitele între care se află datele: câte valori sunt citite, ce semn au, între ce margini se încadrează. Din restricții se obțin cazurile-limită pe care programul este testat.

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 subprobleme, 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

Problema este descompusă în subprobleme:

1. citirea celor n sume, una câte una;
2. adunarea sumelor pozitive, care dau totalul încasărilor;
3. adunarea sumelor negative, luate cu semn schimbat, care dau totalul cheltuielilor;
4. scăderea celor două totaluri, care dă soldul.

Prima subproblemă cere o structură repetitivă, iar a doua și a treia sunt rezolvate în interiorul aceleiași parcurgeri: la fiecare sumă citită, semnul ei 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 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)

```

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

Programul următor trece testul cu date numai pozitive, dar dă rezultate greșite la celelalte:

```
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)
```

Pentru datele `3 | 120 -50 -30` se obține `120, -80, 200`, în loc de `120, 80, 40`. Cauza se află pe ramura `else`: sumele negative sunt adunate ca atare, deci variabila `cheltuieli` primește o valoare negativă. Corectarea înseamnă înlocuirea semnului `+` cu `-`, iar programul este apoi testat din nou, pe toate seturile de date.

5. Activitate practică

Pentru fiecare problemă se stabilesc mai întâi datele de intrare, de ieșire și de manevră, apoi se scrie algoritmul și abia la urmă programul. Programul se poate executa direct în pagină, în caseta de la finalul secțiunii.

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.

```
n = int(input())

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

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

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.

```
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)
```

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 5.0

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 nu poate fi făcută.

```
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")
```

6. 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.
- Testarea se face pe seturi de date pentru care rezultatul este cunoscut dinainte, inclusiv pe cazuri-limită.
- Erorile descoperite la testare duc la reluarea etapelor anterioare, nu la corectarea programului la întâmplare.