INFORMATICĂ · CLASA A IX-A · ALGORITMI · LECȚIA 1

Algoritmi, date și pseudocod

Durată: 60 min Nivel: toate profilurile

CE VEI ȘTI DUPĂ LECȚIE

- Explici ce este un algoritm și ce proprietăți trebuie să aibă.
- Deosebești datele de intrare, de ieșire și de manevră (variabile, constante).
- Calculezi expresii cu operatorii folosiți la BAC: [ ] (partea întreagă), % (restul), operatori logici.
- Citești și scrii pseudocod cu cele trei structuri: liniară, alternativă, repetitivă.
- Urmărești execuția unui algoritm într-un tabel de valori, exact cum se cere la BAC.

1. Ce este un algoritm?

Un **algoritm** este o succesiune **finită** de pași **clari** care, pornind de la niște date, ajunge la un rezultat. O rețetă de clătite, instrucțiunile de montare ale unui dulap sau drumul de acasă la liceu, explicat unui prieten, sunt algoritmi din viața reală.

În informatică, algoritmul este **planul**. Programul în C++ este doar traducerea acestui plan într-o limbă pe care o înțelege calculatorul. De aceea, la BAC, un întreg subiect (al II-lea) se dă în **pseudocod**: o limbă intermediară, în română, care nu depinde de limbajul de programare.

Proprietățile unui algoritm

Proprietate	Ce înseamnă	Exemplu care o încalcă
Claritate (determinism)	fiecare pas e descris exact, fără ambiguități	„pune *puțină* sare”
Finitudine	se termină după un număr finit de pași	„cât timp 1 = 1, scrie 5” (nu se oprește niciodată)
Generalitate	rezolvă o *clasă* de probleme, nu un singur caz	un algoritm care calculează doar 3 + 5
Eficiență	folosește cât mai puțin timp și memorie	numărarea divizorilor lui n mergând până la n în loc de √n

DE ȘTIUT

La BAC apar des întrebări de tipul „care dintre secvențe *nu* se termină?”. Verifică două lucruri: variabila din condiție **se modifică** în buclă? Și se modifică **spre** condiția de oprire? De exemplu, `cât timp x > 0 execută x ← x + 1` nu se oprește niciodată pentru x = 1, pentru că x crește, deci se îndepărtează de oprire.

2. Datele cu care lucrează un algoritm

- **Date de intrare**: ce primim (le *citim*). De exemplu numărul `n`.

- **Date de ieșire**: rezultatul (îl *scriem*). De exemplu suma cifrelor lui `n`.
- **Date de manevră** (auxiliare): variabile ajutătoare, ca un contor sau o sumă parțială.

O **variabilă** este o cutie cu nume în care păstrăm o valoare care se poate schimba. O **constantă** are o valoare fixă: `10`, `3.14`, `'a'`.

Operația de bază este **atribuirea**, notată în pseudocod cu `←`:

```
x ← 5        // în cutia x se pune valoarea 5
x ← x + 1    // se ia valoarea veche a lui x, se adună 1, rezultatul se pune înapoi în x: x
devine 6
```



Atribuirea x ← x + 1: întâi se calculează partea dreaptă cu valoarea veche, apoi rezultatul înlocuiește conținutul cutiei

ATENȚIE — GREȘELI FRECVENTE

`x ← x + 1` **nu** este o ecuație (ca ecuație n-ar avea soluție). Se citește „x primește x plus 1": întâi se calculează partea din dreapta, cu valoarea **veche** a lui x, apoi rezultatul se pune în stânga, iar valoarea veche se pierde.

Exemplu rezolvat: interschimbarea a două valori

Vrem ca `a` și `b` să își schimbe valorile. Varianta naivă nu merge:

```
a ← b      // a primește valoarea lui b; valoarea veche a lui a s-a pierdut!
b ← a      // b primește tot valoarea lui b
```

Pentru `a = 3, b = 7`: după primul pas a = 7, după al doilea b = 7. Ambele sunt 7. Soluția este o **a treia cutie**, `aux`, în care salvăm valoarea care ar urma să se piardă:

```
aux ← a    // aux = 3
a ← b      // a = 7
b ← aux    // b = 3
```

3. Expresii și operatori

Operatori aritmetici

Operator	Semnificație	Exemplu	Rezultat
`+ - *`	adunare, scădere, înmulțire	`7 * 3`	21
`/`	împărțire (rezultat real, în pseudocod)	`7 / 2`	3.5
`[x]`	partea întreagă a lui x	`[7 / 2]`	3

Operator	Semnificație	Exemplu	Rezultat
`%`	restul împărțirii întregi	`7 % 2`	1

Cum calculezi pe hârtie `a % b` și `[a / b]` pentru numere naturale: faci împărțirea cu rest de la matematică. `17 : 5 = 3 rest 2`, deci `[17 / 5] = 3` și `17 % 5 = 2`. Dacă `a < b`, câtul e 0 și restul e chiar `a`: `[5 / 17] = 0`, `5 % 17 = 5`.

Trucurile care apar în aproape orice problemă cu numere:

- `n % 10` este **ultima cifră** a lui n (`1234 % 10 = 4`);
- `[n / 10]` este n **fără ultima cifră** (`[1234 / 10] = 123`);
- `n % 2 = 0` înseamnă că n este **par**;
- `a % b = 0` înseamnă că **b divide pe a** (a se împarte exact la b);
- `n % 100` dă ultimele două cifre; `[n / 100] % 10` dă cifra sutelor.

Operatori relaționali și logici

- relaționali: `< ≤ > ≥ = ≠`; rezultatul este **adevărat** sau **fals**;
- logici: **și** (ambele adevărate), **sau** (măcar una adevărată), **not** (negația).

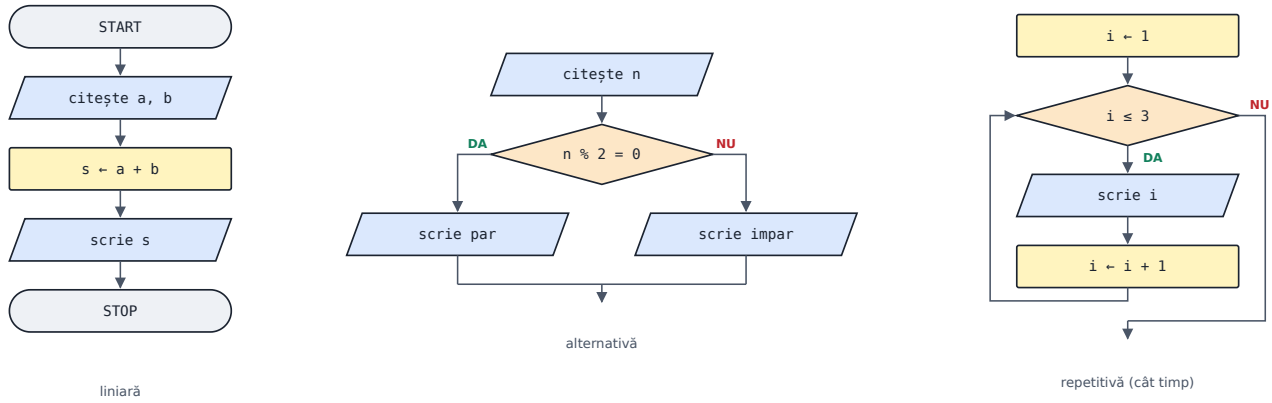
a	b	a și b	a sau b	not a
A	A	A	A	F
A	F	F	A	F
F	A	F	A	A
F	F	F	F	A

> **LA BAC**
>
> Legile lui De Morgan apar foarte des la grilele de BAC:
>
> - **not (a și b) = (not a) sau (not b)**
> - **not (a sau b) = (not a) și (not b)**
>
> Exemplu: „x NU aparține intervalului [10, 20]" înseamnă `not (x ≥ 10 și x ≤ 20)`, adică `x < 10 sau x > 20`. Observă că și relațiile se neagă: `≥` devine `<`, `≤` devine `>`.

Ordinea operațiilor: întâi parantezele, apoi `not`, apoi `* / %`, apoi `+ -`, apoi relaționalii, apoi `și`, apoi `sau`. Dacă ai dubii, pune paranteze: nu greșești niciodată punând paranteze în plus.

4. Structurile de bază ale pseudocodului

Orice algoritm se poate scrie folosind doar **trei structuri**. Aceasta e ideea programării structurate și o vei regăsi în orice limbaj.



Cele trei structuri ca scheme logice: liniară (pași unul după altul), alternativă (o decizie cu două ramuri), repetitivă (un grup de pași care se repetă cât timp o condiție e adevărată)

4.1 Structura liniară (secvențială)

Pașii se execută unul după altul, de sus în jos.

```
citește a, b
s ← a + b
p ← a * b
scrie s, " ", p
```

4.2 Structura alternativă (decizia)

```
┌dacă n % 2 = 0 atunci
│   scrie "par"
│altfel
│   scrie "impar"
└■
```

Ramura `altfel` poate lipsi: atunci, dacă condiția e falsă, nu se întâmplă nimic și se trece mai departe.

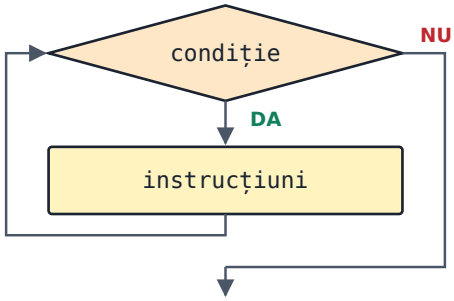
4.3 Structurile repetitive

Avem trei variante și trebuie să le recunoști pe toate:

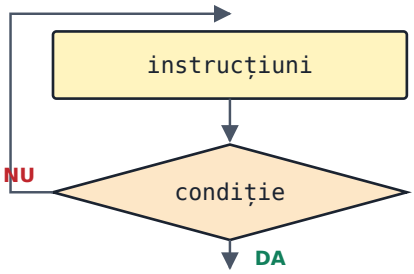
```
┌cât timp condiție execută        // testul este ÎNAINTE: se poate să nu execute nimic
│   instrucțiuni
└■

┌repetă                           // testul este DUPĂ: execută cel puțin o dată
│   instrucțiuni
└până când condiție               // se OPREȘTE când condiția devine adevărată

┌pentru i ← 1, n execută          // i ia pe rând valorile 1, 2, ..., n
│   instrucțiuni
└■
```



cât timp: testează întâi



repetă ... până când: execută întâi

cât timp testează condiția înainte de corp și se oprește când e falsă; repetă ... până când execută corpul o dată, apoi se oprește când condiția devine adevărată

CAPCANĂ

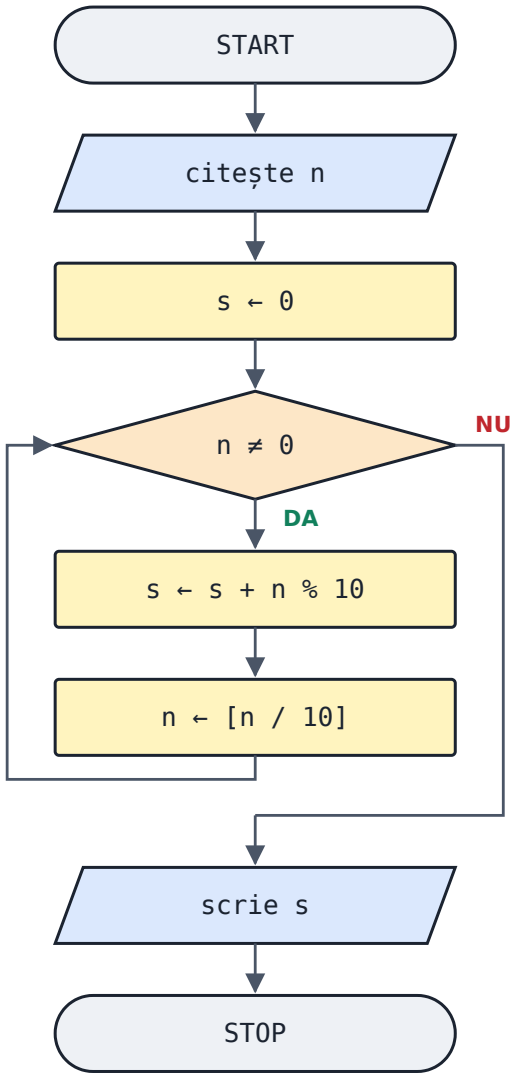
`cât timp` **continuă** cât timp condiția e adevărată, dar `repetă … până când` **se oprește** când condiția devine adevărată. Când transformi una în alta, condiția se **neagă**: `cât timp n > 0` corespunde lui `până când n ≤ 0` (cu grijă la cazul în care bucla nu trebuie să se execute deloc).

5. Primul algoritm complet: suma cifrelor

Problema: se citește un număr natural `n`. Afișează suma cifrelor lui.

Ideea, înainte de orice pseudocod. Cum ai face de mână pentru 472? Te uiți la ultima cifră (2), o aduni; o „ștergi”; te uiți la noua ultimă cifră (7), o aduni; și tot așa până nu mai rămâne nimic. Avem deja tot ce ne trebuie: ultima cifră e `n % 10`, iar „ștergerea” ei e `n ← [n / 10]`. Ne oprim când n devine 0.

```
citește n                    (număr natural)
s ← 0
┌cât timp n ≠ 0 execută
│   s ← s + n % 10
│   n ← [n / 10]
└■
scrie s
```



Schema logică a algoritmului pentru suma cifrelor

Urmărirea execuției (tabelul de valori)

Pentru `n = 472`. Fiecare rând este o trecere prin buclă; coloanele arată valorile **după** pasul respectiv.

Pas	`n ≠ 0 ?`	`n % 10`	`s`	`n`
start	–	–	0	472
1	da	2	2	47
2	da	7	9	4
3	da	4	13	0
4	**nu**, ieșim	–	13	0

Se afișează **13**.

SFAT

La BAC, la subiectul al II-lea, primești un algoritm în pseudocod și trebuie să spui ce afișează pentru anumite valori. **Fă mereu tabelul de valori pe ciornă**, câte o coloană pentru fiecare variabilă. E metoda care nu dă greș și te ferește de greșelile de neatenție.

DE CE AȘA?

De ce `s ← 0` înainte de buclă și nu în buclă? Dacă l-am pune în buclă, la fiecare pas suma s-ar reseta și ar rămâne doar ultima cifră adunată.

De ce `cât timp` și nu `pentru`? Nu știm de la început câte cifre are numărul, deci nu știm de câte ori se repetă bucla. `pentru` e potrivit când știm numărul de repetări.

Și dacă n = 0?

Condiția `n ≠ 0` e falsă de la început, bucla nu se execută, se afișează `s = 0`. Corect: suma cifrelor lui 0 este 0. Dar dacă am număra cifrele cu același șablon, am obține 0 cifre pentru numărul 0, ceea ce e greșit (0 are o cifră). Vei rezolva asta în exercițiile de mai jos.

6. Schema logică, pe scurt

Bloc	Formă	Rol
START / STOP	oval	începutul și sfârșitul
Citire / scriere	paralelogram	`citește n`, `scrie s`
Calcul / atribuire	dreptunghi	`s ← s + n % 10`
Decizie	romb	`n ≠ 0 ?` cu două ieșiri: DA / NU

La liceu se lucrează aproape doar cu pseudocod, dar e bine să recunoști formele: apar în manuale și la gimnaziu.

7. Greșeli frecvente

ATENȚIE — GREȘELI FRECVENTE

> - Confunzi `/` (împărțire reală) cu `[ / ]` (câtul întreg): `[7 / 2] = 3`, dar `7 / 2 = 3.5`. - Calculezi `%` la numere negative „după ureche”. La BAC apar aproape mereu numere naturale; ține-te de împărțirea cu rest. - Uiți că `repetă … până când` se oprește când condiția e **adevărată**. - Negi greșit o condiție compusă: negația lui `x ≥ 10 și x ≤ 20` nu este `x < 10 și x > 20` (imposibil), ci `x < 10 sau x > 20`. - În tabelul de valori, scrii valoarea nouă a unei variabile înainte de a o fi folosit pe cea veche.

8. La BAC

> **LA BAC**
>
> Subiectul I conține grile cu expresii: „care este valoarea expresiei…", „care expresie are valoarea 1 dacă și numai dacă…". Subiectul al II-lea începe aproape mereu cu un algoritm în pseudocod: (a) ce afișează pentru niște valori date, (b) găsește valori pentru care afișează un rezultat dat, (c) scrie programul C++ corespunzător, (d) scrie un algoritm echivalent cu altă structură repetitivă. Tot ce ai văzut în lecția asta e baza acestor cerințe.

9. Exersează pe hârtie

Răspunsurile sunt la finalul lecției, după probleme: încearcă întâi singur.

1. Calculează: `[17 / 5]`, `17 % 5`, `[5 / 17]`, `5 % 17`, `[1234 / 100] % 10`.

2. Care expresie este adevărată dacă și numai dacă numărul natural `x` are **exact două cifre**?

a) `x ≥ 10 sau x ≤ 99`
b) `x ≥ 10 și x ≤ 99`
c) `x > 10 și x < 99`
d) `x % 100 = x`

3. Care este negația expresiei `(a > 0) și (b ≤ 5)`?

a) `(a ≤ 0) și (b > 5)`
b) `(a < 0) sau (b > 5)`
c) `(a ≤ 0) sau (b > 5)`
d) `(a > 0) sau (b ≤ 5)`

4. Ce afișează algoritmul din secțiunea 5 pentru `n = 9081`? Dar pentru `n = 1000`?

5. Pentru care dintre valori secvența de mai jos **nu** se termină?

```
┌cât timp x ≠ 10 execută
│   x ← x + 3
└■
```

a) x = 1 b) x = 4 c) x = 7 d) x = 2

6. De câte ori se execută `scrie "*"` în secvența următoare?

```
i ← 5
┌repetă
│   scrie "*"
│   i ← i + 1
└până când i > 3
```

7. Modifică algoritmul din secțiunea 5 ca să calculeze **numărul de cifre** al lui `n`, corect și pentru `n = 0`.

8. Scrie în pseudocod un algoritm care citește trei numere și îl afișează pe cel mai mare.

10. Mai departe (pentru cine vrea mai mult)

Același rezultat, algoritmi diferiți. Suma numerelor 1 + 2 + ... + n se poate calcula cu o buclă (n pași) sau cu formula lui Gauss, `n(n + 1) / 2` (un singur pas). Pentru n = 1 000 000, prima variantă face un milion de adunări, a doua o înmulțire și o împărțire. Asta înseamnă **eficiență**: aceeași problemă, mult mai puțin de lucru. Vei învăța să măsori asta în lecția IX.14.

Algoritmii sunt vechi. Algoritmul lui Euclid pentru cel mai mare divizor comun are peste 2 300 de ani și e folosit și azi, de exemplu în criptografia care protejează plățile online. Cuvântul „algoritm" vine de la numele matematicianului persan al-Khwarizmi (secolul IX).

Unde întâlnești algoritmi în fiecare zi. Aplicația de hărți caută cel mai scurt drum (un algoritm pe grafuri, lecția XI.19), un magazin online îți ordonează produsele după preț (sortare, lecția X.4), iar căutarea într-o listă de contacte folosește căutarea binară (lecția X.5). Tot ce vom face în anii de liceu are aplicații concrete.

Rezumat

- Algoritm = pași clari, finiți, generali și cât mai eficienți.
- Atribuirea `x ← expresie` calculează întâi dreapta, apoi înlocuiește valoarea din x.
- `n % 10` = ultima cifră; `[n / 10]` = taie ultima cifră; `a % b = 0` = b divide pe a.
- Trei structuri: liniară, alternativă (`dacă`), repetitivă (`cât timp` , `repetă … până când` , `pentru`).
- Tabelul de valori este cea mai bună unealtă pentru exercițiile de tip „ce afișează?".

Temă

1. Fă-ți cont pe **pbinfo.ro** (gratuit): de la lecția următoare vei trimite acolo programe.
2. Rezolvă pe caiet exercițiile 4–8 de mai sus, fără să te uiți la răspuns.
3. Scrie în pseudocod: se citesc `n` numere; afișează câte dintre ele sunt pare. Fă tabelul de valori pentru n = 4 și numerele 3, 8, 10, 7.
4. Rezolvă grilele de BAC de mai jos, pe pbinfo.

Grile din subiectele de BAC (pbinfo)

Exerciții de tip grilă din variantele de Bacalaureat, rezolvate online pe pbinfo, cu verificare automată a răspunsului.

- [ ] `#31` **Exercițiul #31**
  completarea unui algoritm în pseudocod (Bac 2009)
  https://www.pbinfo.ro/?pagina=itemi-evaluare&id=31

- [ ] `#1405` **Exercițiul #1405**
  algoritm în pseudocod: ce afișează (Bac 2023)
  https://www.pbinfo.ro/?pagina=itemi-evaluare&id=1405

- [ ] `#1406` **Exercițiul #1406**
  același algoritm: valori pentru care afișează un rezultat dat
  https://www.pbinfo.ro/?pagina=itemi-evaluare&id=1406

- [ ] #10 **Exercițiul #10**
  evaluarea unei expresii cu / întreg (Bac 2009)
  https://www.pbinfo.ro/?pagina=itemi-evaluare&id=10

- [ ] #15 **Exercițiul #15**
  expresii logice echivalente (De Morgan)
  https://www.pbinfo.ro/?pagina=itemi-evaluare&id=15

- [ ] #33 **Exercițiul #33**
  ce valori poate avea o expresie cu %
  https://www.pbinfo.ro/?pagina=itemi-evaluare&id=33

Citește și exersează în plus

- Informatică, clasa a IX-a — toate resursele https://www.pbinfo.ro/articole/5547/informatica-clasa-a-ix-a
- Eficiența algoritmilor https://www.pbinfo.ro/articole/39626/eficienta-algoritmilor
- Algopedia — cercul de informatică pentru gimnaziu — lecții de algoritmică pornind de la zero, cu multe exerciții de gândire https://www.algopedia.ro/wiki/index.php/Cercul_de_informatic%C4%83_gimnaziu

Răspunsuri

Exersează pe hârtie

1. 3, 2, 0, 5, 2. Pentru ultima: `[1234 / 100] = 12`, iar `12 % 10 = 2` (cifra sutelor lui 1234).

2. b) Varianta a) e adevărată pentru orice număr. Varianta c) exclude 10 și 99, care au două cifre. Varianta d) e adevărată și pentru numerele de o cifră (de exemplu 5 % 100 = 5).

3. c) După De Morgan, „și" devine „sau", iar fiecare relație se neagă: negația lui `a > 0` este `a ≤ 0` (nu `a < 0`, care uită cazul a = 0).

4. 18 (9 + 0 + 8 + 1) și 1 (1 + 0 + 0 + 0). Bucla se execută de 4 ori în ambele cazuri: zerourile sunt cifre și sunt parcurse.

5. d) Pornind de la 2, x ia valorile 5, 8, 11, 14…: sare peste 10 și nu va fi niciodată egal cu 10. Pentru 1, 4 și 7 ajunge exact la 10 (1 → 4 → 7 → 10).

6. O dată. `repetă` execută corpul înainte de a testa: se scrie o steluță, i devine 6, iar `6 > 3` e adevărat, deci bucla se oprește. Cu `cât timp i ≤ 3` corpul nu s-ar fi executat deloc: aceasta e diferența dintre cele două structuri.

7. Cea mai simplă soluție folosește `repetă`, care execută corpul cel puțin o dată:

```
citește n
nr ← 0
┌repetă
│   nr ← nr + 1
│   n ← [n / 10]
└până când n = 0
scrie nr
```

Pentru n = 0: nr devine 1, n rămâne 0, iar condiția `n = 0` oprește bucla. Se afișează 1.

8.

```
citește a, b, c
max ← a
┌dacă b > max atunci max ← b
└■
┌dacă c > max atunci max ← c
└■
scrie max
```

Ideea: presupunem că primul e cel mai mare, apoi îl comparăm pe fiecare dintre celelalte cu „campionul" de până atunci. Aceeași idee o vei folosi pentru maximul a n numere.