

ALGORITMI

Cuvântul algoritm provine de la numele unui matematician arab (Mohammed ibn-Musa al-Khowarizmi) a cărui lucrări au fost traduse în latina sub numele de **Algorismus**.

Definiție: Un algoritm este o secvență finită de operații, ordonată și bine definită, care pe baza datelor de intrare, conduce la obținerea datelor de ieșire (rezultate).

Proprietățile algoritmilor

a) **Claritatea** – la fiecare moment, operația care urmează a fi executată este unic determinată, definită și realizabilă (adică poate fi efectuată la momentul respectiv, cu mijloacele disponibile).

De exemplu, secvența “Dacă plouă stau acasă sau merg la cinema” nu este clară, deoarece, în cazul în care nu plouă, operația care se execută nu este unic determinată. Sau să presupunem că dorim să obținem un număr natural, care se poate scrie ca sumă de pătrate. Secvența de mai jos “scrie $x^2 + y^2$ ” nu este clară deoarece nu putem calcula valoarea $x^2 + y^2$, deoarece nu cunoaștem valorile lui x și y .

b) **Generalitatea** (universalitatea) – o secvență de pași reprezintă un algoritm de rezolvare a unei probleme dacă obține date de ieșire (rezultate) pentru orice date de intrare specifice problemei.

Secvența de pași prezentată în exemplul 2 este generală, deoarece conduce la rezolvarea ecuației $ax + b = 0$ pentru orice valori reale ale coeficienților a și b . Dar, dacă am fi descris o secvență de pași care să rezolve numai ecuația $x + 2 = 0$, aceasta nu ar fi fost un algoritm !

c) **Finititudine** – rezultatele problemei se obțin după un număr finit de pași.

De exemplu, problema “Să se determine toate zecimalele numărului ” nu are o soluție algoritmică, deoarece este un număr irațional, care are o infinitate de zecimale. Dar dacă am enunța problema astfel : “Fie n un număr natural dat. Să se determine primele n zecimale ale numărului ”, această problemă admite o soluție algoritmică, deoarece primele n zecimale se pot obține după un număr finit de pași.

d) **Eficiența** – în situația în care în rezolvarea unei probleme avem de ales între mai mulți algoritmi, îl vom folosi pe acela care are un timp de lucru mai mic asigurând astfel “eficiența din punct de vedere al timpului”. Dacă algoritmii folosesc spațiu diferit de memorie, atunci îl vom folosi pe acela care folosește cât mai puțină memorie asigurând astfel “eficiența din punct de vedere al spațiului de memorie”.

Reprezentarea algoritmilor

- În limbaj natural
- Cu ajutorul schemelor logice
- Cu ajutorul limbajului pseudocod
- Cu ajutorul limbajelor de programare

Limbajul pseudocod:

Operatori:

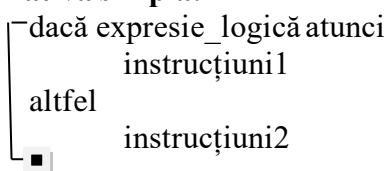
- Aritmetici: $+$, $-$, $*$, $/$, $\%$
- Relaționali: $=$, \neq , $<$, $>$, \leq , \geq
- Logici: și, sau, not

Instrucțiuni

1. citește variabilă1, variabilă2, ...
2. scrie expresie1, expresie2 (mesajele se precizează între perechi de ghilimele)
3. **Atribuirea:** variabilă ← expresie

Structuri:

4. Alternativă simplă:



```

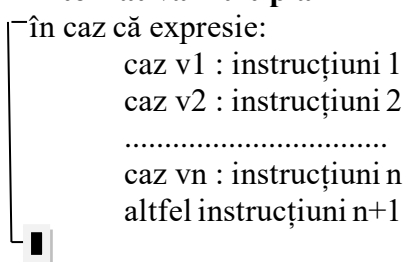
if(expresie_logică)
    { instrucțiuni 1;
    }
else
    { instrucțiuni2;
    }

```

Observații: ramura altfel nu este obligatorie.

Efectul instrucțiunii: se evaluează expresia logică, dacă este adevărată se execută secvența de instrucțiuni1, altfel se execută secvența de instrucțiuni2 sau nimic dacă ramura altfel lipsește. După executarea unstrucțiunii ”dacă”, algoritmul continuă cu instrucțiunile care urmează după aceasta.

5. Alternativă multiplă



```

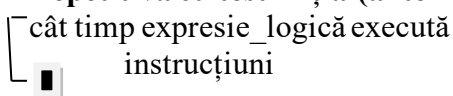
switch(expresie)
{ case v1 : instrucțiuni 1;break;
  case v2 : instrucțiuni 2;break;
  .....
  case vn : instrucțiuni n;break;
  default instrucțiuni n+1;
}

```

Observații: ramura altfel nu este obligatorie.

Efectul instrucțiunii: se evaluează expresia și dacă valoarea acesteia este egală cu v1 se execută secvența de instrucțiuni 1, dacă valoarea acesteia este egală cu v2 se execută secvența de instrucțiuni v2,....., dacă valoarea acesteia este egală cu vn se execută secvența de instrucțiuni n, iar dacă nu este egală cu nici una din valorile v1,...,vn, se execută secvența instrucțiuni n+1 sau nimic dacă ramura altfel lipsește.

6. Repetitivă cu test inițial(anterior condiționată)



```

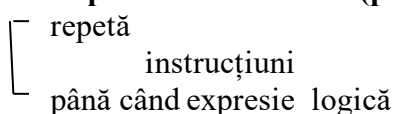
while (expresie_logică)
{
    instrucțiuni;
}

```

Efectul instrucțiunii: se evaluează expresia logică și dacă rezultatul este adevărat se execută secvența de instrucțiuni și apoi algoritmul revine la evaluarea expresiei logice. Dacă la o evaluare expresia logică este falsă algoritmul continuă cu instrucțiunile care urmează după instrucțiunea cât timp.

Observații: numărul minim posibil de execuții al secvenței de instrucțiuni este 0, dacă de la început expresia logică este falsă

7. Repetitivă cu test final(posterior condiționată)



```

do
{
    instrucțiuni;
}
while ! (expresie_logică);

```

Efectul instrucțiunii: se execută secvența de instrucțiuni și apoi se evaluează expresia logică și dacă rezultatul este fals se revine la executarea secvenței de instrucțiuni. Dacă la o evaluare expresia logică este adevărată, algoritmul continuă cu instrucțiunile care urmează după instrucțiunea ”repetă”.

Variantă de exprimare:

┌ execută
│ instrucțiuni
└ cât timp expresie_logică

```
do
{
    instrucțiuni;
}
while (expresie_logică);
```

Observații: numărul minim posibil de execuții al secvenței de instrucțiuni este 1.

8. Repetitivă cu număr cunoscut de pași(care folosește contor)

┌ pentru $c \leftarrow$ valoare_inițială, valoare_finală, pas execută
│ instrucțiuni
└ ■

```
for(c=valoare_inițială;c<=valoare_finală,c=c+pas)
{
    instrucțiuni;
}
```

Efectul instrucțiunii: se inițializează contorul c cu valoarea_inițială, apoi se compară contorul cu valoarea_finală, și dacă nu a fost depășită, se execută secvența de instrucțiuni, după care contorul se modifică cu valoarea pasului($c \leftarrow c + \text{pas}$). Dacă contorul depășește valoarea_finală algoritmul continuă cu instrucțiunile care urmează după instrucțiunea pentru.

Observații:

- numărul de execuții al secvenței de instrucțiuni este $\lfloor \text{valoare_finală} - \text{valoare_inițială} / \text{pas} \rfloor + 1$
- dacă pasul este 1 acesta poate să nu fie scris.

Numim **iterație** o execuție a unei secvențe de instrucțiuni din cadrul unei instrucțiuni repetitive.

Transformări între structurile repetitive:

1. din cât timp în repetă/execută

┌ cât timp expresie_logică execută
│ instrucțiuni
└ ■

se transformă în:

┌ dacă expresie_logică atunci
│ ┌ repetă
│ │ instrucțiuni
│ └ până când **not**(expresie_logică)
└ ■

sau

┌ dacă expresie_logică atunci
│ ┌ execută
│ │ instrucțiuni
│ └ cât timp(expresie_logică)
└ ■

deoarece dacă de la început expresia logică din cât timp nu este adevărată, secvența de instrucțiuni nu trebuie să se execute niciodată nici în secvența repetă/execută.

2. din repetă/execută în cât timp

```
┌ repetă
│   instrucțiuni
└ până când expresie_logică
se transformă în:
    instrucțiuni
┌ cât timp not(expresie_logică) execută
│   instrucțiuni
└ ■
```

sau

```
┌ execută
│   instrucțiuni
└ cât timp expresie_logică
se transformă în:
    instrucțiuni
┌ cât timp (expresie_logică) execută
│   instrucțiuni
└ ■
```

deoarece secvența de instrucțiuni trebuie să se execute cel puțin o dată.

3. din pentru în cât timp

```
┌ pentru c←valoare_inițială, valoare_finală, pas execută
│   instrucțiuni
└ ■
se transformă în:
    c←valoare_inițială
┌ cât timp c<=valoare_finală execută
│   instrucțiuni
│   c←c+pas
└ ■
```

dacă pasul este negativ vom scrie $c >= \text{valoare_finală}$

4. din pentru în repetă/execută

```
┌ pentru c←valoare_inițială, valoare_finală, pas execută
│   instrucțiuni
└ ■
se transformă în:
    c←valoare_inițială
┌ dacă c<=valoare_finală atunci
│   ┌ repetă
│   │   instrucțiuni
│   │   c←c+pas
│   └ până când c>valoare_finală
```

dacă pasul este negativ vom scrie dacă $c > \text{valoare_finală}$...respectiv până când $c < \text{valoare_finală}$

sau se transformă în

```

c ← valoare_inicială
dacă c ≤ valoare_finală atunci
    execută
        instrucțiuni
        c ← c + pas
    cât timp c ≤ valoare_finală

```

dacă pasul este negativ vom scrie dacă $c \leq \text{valoare_finală}$...respectiv cât timp $c \geq \text{valoare_finală}$

5. din cât timp sau repetă/execută în pentru putem transforma doar dacă recunoaștem o variabilă pe post de contor care să primească o valoare inițială, să se compare cu o valoare finală și să se modifice cu un pas constant

```

c ← valoare_inicială
cât timp c ≤ valoare_finală execută (c ≥ valoare_finală pentru pas negativ)
    instrucțiuni
    c ← c + pas

```

respectiv

```

c ← valoare_inicială
repetă
    instrucțiuni
    c ← c + pas
până când c > valoare_finală (c < valoare_finală pentru pas negativ)

```

sau

```

c ← valoare_inicială
execută
    instrucțiuni
    c ← c + pas
cât timp c ≤ valoare_finală (c ≥ valoare_finală pentru pas negativ)

```

se transformă în:

```

pentru c ← valoare_inicială, valoare_finală, pas execută
    instrucțiuni

```

Observații:

1. Există probleme când expresia logică din cât timp(repetă) este formată din două expresii unite printr-un operator logic(de cele mai multe ori "și") cerința fiind transformarea în structura "pentru".

În acest caz în secvența de instrucțiuni se va adăuga o condiție care să testeze cazul în care se iese forțat din instrucțiune.

Exemplu:

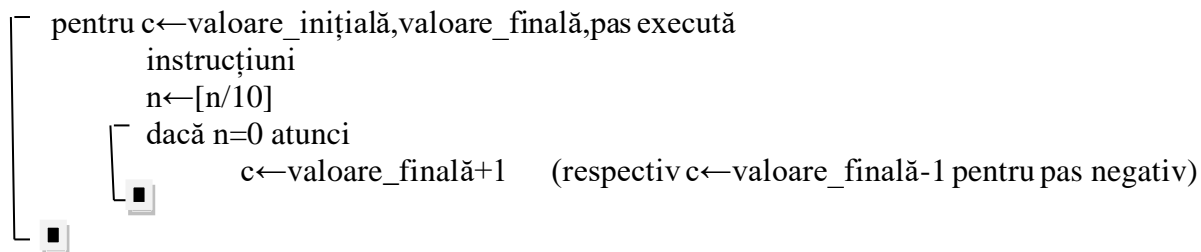
```

citește n
c ← valoare_inicială
cât timp c ≤ valoare_finală și n > 0 execută
    instrucțiuni
    n ← [n/10]
    c ← c + pas

```

se transformă în:

citeste n



2. Uneori se cere transcrierea algoritmului **fără a folosi structuri repetitive**, sau eliminând una dinre ele dacă sunt mai multe. În acest caz structurile repetitive de eliminat se vor înlocui cu formule, teste simple folosind instrucțiuni alternative.

(2009 varianta bacalaureat 93-94,96-100)

3. NU SE VA FACE NICIODATĂ TRANSFORMAREA ÎNTRE STRUCTURILE REPETĂ ȘI EXECUTĂ DEOARECE AMBELE FAC PARTE DIN ACEEAȘI CATEGORIE DE STRUCTURI REPETITIVE CU TEST FINAL (ȘI DE OBICEI SE CERE TRANSFORMAREA ÎN ALT TIP DE STRUCTURĂ REPETITIVĂ(CU TEST ÎNIȚIAL SAU NUMĂR CUNOSCUT DE PAȘI))

4. **Atenție la prioritățile operatorilor** ”și” are prioritate mai mare decât ”sau”, dar are prioritate mai mică decât ”not”.....

PROBLEME LIMBAJ PSEUDOCOD:

1. Se consideră algoritmul alăturat, descris în pseudocod.

S-a notat cu $a \% b$ restul împărțirii numărului natural

a la numărul natural nenul b , iar cu $[a/b]$ câtul împărțirii întregi a numărului natural a la numărul natural nenul b .

a) Scrieți numărul care se va afișa dacă se citește pentru n valoarea **528791** și pentru k valoarea **6**. (6p.)

b) Dacă pentru k se citește valoarea **9** scrieți toate valorile formate din exact **5** cifre care se pot citi pentru variabila n , astfel încât rezultatul afișat să fie, de fiecare dată, **2008**. (6p.)

c) Scrieți programul C/C++ corespunzător algoritmului dat. (10p.)

d) Scrieți în pseudocod un algoritm echivalent cu cel dat care să utilizeze în locul structurii **cât timp...execută** o structură repetitivă condiționată posterior. (4p.)
(2009 varianta 95)

```
x ← 0
citește n, k
(numere naturale nenule)
cât timp n ≠ 0 execută
    dacă n % 10 < k atunci
        x ← x * 10 + n % 10
    n ← [n / 10]
scrie x
```

2. Se consideră algoritmul alăturat, descris în pseudocod.

S-a notat cu $[x]$ partea întreagă a numărului real x .

a) Scrieți ce se afișează dacă se citește, în această ordine, valorile: **5, 8, 12, 15, 10, 25, 9, 8, 30, 10**. (6p.)

b) Dacă pentru n se citește valoarea **3** scrieți un șir de date de intrare astfel încât ultima valoare care se afișează să fie **3**. (4p.)

c) Scrieți programul C/C++ corespunzător algoritmului dat. (10p.)

d) Scrieți un algoritm pseudocod echivalent cu cel dat în care structura **repetă...până când** să fie înlocuită cu o structură repetitivă cu test inițial. (6p.)
(2009 varianta 92)

```
citește n
(număr natural nenul)
nr ← 0
y ← 0
pentru i ← 1, n execută
    repetă
        citește x (număr real)
        nr ← nr + 1
        până când x >= 1 și x <= 10
    y ← y + x
scrie [y / n]
scrie nr
```

3. Se consideră algoritmul alăturat descris în pseudocod.

S-a notat cu $[a]$ partea întreagă a numărului real a și cu $|b|$ valoarea absolută a numărului întreg b .

a) Scrieți valoarea care se va afișa pentru $z=50$ și $x=1$. (6p.)

b) Scrieți în pseudocod un algoritm echivalent cu cel dat, în care să se înlocuiască structura **repetă...până când** cu o structură repetitivă cu test inițial. (6p.)

c) Scrieți programul C/C++ corespunzător algoritmului dat. (10p.)

d) Dacă pentru z se citește numărul **30**, scrieți o valoare care, citită pentru x , determină ca atribuirea $y \leftarrow x$ să se execute o singură dată. (4p.)
(2009 varianta 91)

```
citește z, x
(numere întregi nenule)
z ← |z|
x ← |x|
repetă
    y ← x
    x ← [(x + z / x) / 2]
până când x = y
scrie x
```

4. Se consideră algoritmul alăturat, descris în pseudocod.

S-a notat cu $x\%y$ restul împărțirii numărului natural x la numărul natural nenul y .

- a) Scrieți valorile care se vor afișa în urma executării algoritmului dacă se citește numerele $a=105$, $b=118$ și $k=7$. (6p.)
b) Dacă pentru k se citește valoarea 7, iar pentru a valoarea 2009, scrieți cea mai mare valoare care se poate citi pentru variabila b , astfel încât numărul afișat să fie -1. (4p.)
c) Scrieți programul C/C++ corespunzător algoritmului dat. (10p.)
d) Scrieți în pseudocod un algoritm echivalent cu cel dat în care să se înlocuiască structura **cât timp...execută** cu o structură repetitivă cu test final. (6p.)

(2009 varianta 90)

```
citește a, b, k
(numere naturale)
t ← a
p ← 0
cât timp t ≤ b execută
    dacă k = t % 10 atunci
        scrie t
        p ← -1
    t ← t + 1
dacă p = 0 atunci
    scrie -1
```

5. Se consideră algoritmul alăturat descris în pseudocod.

- a) Scrieți valoarea care se afișează dacă se citește numerele $n=2$ și $m=11$. (6p.)
b) Scrieți programul C/C++ corespunzător algoritmului dat. (10p.)
c) Dacă pentru n se citește valoarea 1 scrieți numărul de valori naturale nenule de exact o cifră, care pot fi citite pentru variabila m , astfel încât să se afișeze valoarea 0. (6p.)
d) Scrieți în pseudocod un algoritm echivalent cu cel dat, care să **NU** folosească structuri repetitive sau recursive. (4p.)

(2009 varianta 93)

```
citește n, m
(numere naturale, n ≤ m)
s ← 0
cât timp n < m execută
    s ← s + n
    n ← n + 3
dacă n = m atunci
    scrie s + n
    altfel
    scrie 0
```

6. Se consideră algoritmul alăturat, descris în pseudocod.

S-a notat cu $x\%y$ restul împărțirii numărului natural x la numărul natural nenul y , iar cu $[x/y]$ câtul împărțirii întregi a numărului natural x la numărul natural nenul y .

- a) Scrieți ce va afișa algoritmul dacă pentru n se citește valoarea 123611. (6p.)
b) Scrieți **câte** valori naturale distincte, formate din patru cifre fiecare, pot fi citite pentru variabila n , astfel încât, pentru fiecare dintre acestea, valoarea afișată de algoritm să fie divizibilă cu 10. (6p.)
c) Scrieți în pseudocod un algoritm echivalent cu cel dat care să utilizeze o singură structură repetitivă. (4p.)
d) Scrieți programul C/C++ corespunzător algoritmului dat. (10p.)

(2009 varianta 94)

```
citește n (număr natural nenul)
n1 ← 0
n2 ← 0
k1 ← 0
cât timp n ≠ 0 execută
    dacă (n % 10) % 2 = 0 atunci
        n2 ← n2 * 10 + n % 10
    altfel
        n1 ← n1 * 10 + n % 10
    k1 ← k1 + 1
    n ← [n / 10]
p ← 1
pentru i ← 1, k1 execută
    p ← p * 10
x ← n2 * p + n1
scrie x
```

TEMĂ : VARIANTE 85-89, 96-100