



ALGORITMI



1. Competențe	3
2. Etapele rezolvării problemelor	5
3. Noțiunea de algoritm. Caracteristici	11
4. Date cu care lucrează algoritmi	18
5. Operații asupra datelor	22
6. Reprezentarea algoritmilor	29
7. Principiile programării structurate	47
8. Analiza eficienței unui algoritm	96
9. Aplicații	106
10. Bibliografie & webografie	107



Competențe generale

- *identificarea datelor care intervin într-o problemă și a relațiilor dintre acestea*
- *elaborarea algoritmilor de rezolvare a problemelor*
- *aplicarea algoritmilor fundamentali în prelucrarea datelor*

Competențe specifice

- *descrierea coerentă a unei succesiuni de operații prin care se obțin din datele de intrare, datele de ieșire*
- *identificarea tipurilor de date necesare pentru rezolvarea unei probleme (date de intrare, de ieșire, de manevră)*
- *analizarea enunțului unei probleme: identificarea datele de intrare și a datele de ieșire și stabilirea pașilor de rezolvare a problemei*
- *reprezentarea algoritmilor în pseudocod*



- *respectarea principiilor programării structurate în procesul de elaborare a algoritmilor*
- *elaborarea unui algoritm de rezolvare a unor probleme din aria curriculară a specializării*
- *alegerea unui algoritm eficient de rezolvare a unei probleme*



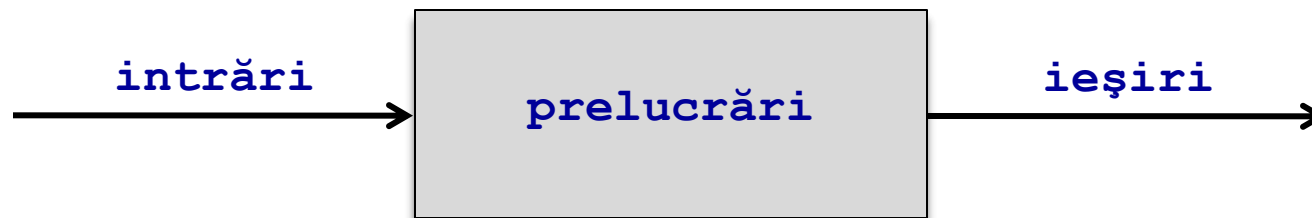
Etapele rezolvării problemelor

gândirea algoritmică – bază a dialogului om-calculator

algoritm – metodă de soluționare a unei probleme

Formalizarea unei metode de rezolvare a unei probleme se poate face numai pe baza unei *gândiri algoritmice*, adică a unui mod de a gândi rezolvarea unei probleme prin prisma rezolvării unei întregi clase de probleme asemănătoare.

Orice prelucrare automată a informațiilor presupune definirea următorului lanț:



Pentru orice rezolvare a unei probleme cu ajutorul calculatorului, trebuie parcurse următoarele etape:

- ① analiza problemei;
- ② elaborarea unui algoritm de rezolvare a problemei;
- ③ implementarea algoritmului într-un limbaj de programare;
- ④ testarea programului și corectarea erorilor.



① *Analiza problemei*

Constă în:

- identificarea funcției programului;
- identificarea fluxului de informații.

Funcția programului determină ceea ce urmează să realizeze programul.

Identificarea fluxului de informații presupune identificarea *informațiilor de intrare* (descrise cu ajutorul datelor de intrare) și a *informațiilor de ieșire* (descrise cu ajutorul datelor de ieșire).



② *Elaborarea unui algoritm de rezolvare a problemei*

Constă în găsirea metodei prin care să se poată rezolva problema.

Presupune identificarea *prelucrărilor* care se fac asupra datelor de intrare pentru a obține datele de ieșire.

Este etapa cea mai importantă și cea mai dificilă, deoarece presupune definirea logică a unei secvențe de operații pe care să le poată executa calculatorul astfel încât să se obțină rezultatele dorite.



③ *Implementarea algoritmului într-un limbaj de programare*

Algoritmul de rezolvare este transpus în limbaj de programare pentru a fi comunicat calculatorului.



④ *Testarea programului și corectarea erorilor*

Testarea programului constă în execuția repetată a programului cu seturi de date de intrare care să prevadă toate situațiile care pot să apară în exploatarea curentă a programului.

Corectarea erorilor de sintaxă și a celor de semantică.

Erorile de sintaxă apar în scrierea incorectă a instrucțiunilor și ele vor fi corectate în program.

Erorile de semantică (logice) apar din cauza metodei de rezolvare alese și ele vor trebui identificate în cadrul programului și corectate în program.



3. Noțiunea de algoritm. Caracteristici

Noțiunea de algoritm. Caracteristici

Pentru a înțelege noțiunea de algoritm vom porni de la un exemplu.

Să presupunem că trebuie să mergem la magazin să cumpărăm un produs. Ce trebuie să facem?



3. Noțiunea de algoritm. Caracteristici

Când am decis să plecăm la magazin vom proceda astfel:

Pasul 1: luăm banii necesari;

Pasul 2: ne îndreptăm către magazin;

Pasul 3: solicităm produsul;

Pasul 4: plătim;

Pasul 5: venim cu produsul acasă.

Am obținut astfel un algoritm:

- care conține 5 etape (deci un număr finit de operații);
- care are scrise etapele în ordinea în care trebuie executate (deci sunt ordonate);
- în care fiecare etapă este explicată în cuvinte (deci este complet definită);
- care pornind de la ceva (în cazul nostru bani) obținem ceea ce dorim (un produs).



În linii mari, relativ la un algoritm putem afirma următoarele:

Un *algoritm* reprezintă o secvență finită de operații, ordonată și complet definită care pornind de la datele de intrare produce rezultatele.



Exemplu

Scrieți un algoritm care calculează și afișează pe ecran suma a două numere întregi **a** și **b**, citite de la tastatură.

Date de intrare: **a, b**

Date de ieșire: **S**



Algoritmul

- Pasul 1. solicită valori pentru **a** și **b**;
- Pasul 2. calculează **S=a+b**;
- Pasul 3. furnizează rezultatul pentru **S**.



Exercițiu

Fiind date trei numere naturale x , y și z , citite de la tastatură, să se determine și să se afișeze pe ecran media aritmetică a acestora.



Caracteristicile algoritmilor

1. **claritatea** – algoritmul să fie precis definit, să prezinte clar și fără ambiguități toate etapele care trebuie parcurse până la obținerea soluției;
2. **finitatea** – algoritmul să fie format dintr-un număr finit de pași, prin executarea cărora să se ajungă la rezolvarea problemei și obținerea rezultatelor;
3. **universalitatea** – algoritmul trebuie să permită rezolvarea unei clase de probleme, care sunt de același tip și care diferă între ele numai prin datele de intrare;
4. **eficiența** – etapele care compun algoritmul trebuie alese astfel încât soluția problemei să fie obținută după un număr minim de pași.



Date cu care lucrează algoritmi

Date

Datele sunt obiecte prelucrate de algoritmi.

Data este un model de reprezentare a algoritmilor, accesibil calculatorului, cu care se poate opera pentru a obține informații.

Din punct de vedere logic, *data* este definită de trei elemente:

- *identificator* - reprezintă un nume format din unul sau mai multe caractere;
- *valoare* – reprezintă conținutul zonei de memorie în care este stocată data;
- *attribute* – reprezintă proprietăți ale datelor care determină modul în care sistemul va trata datele.

Cel mai important atribut este *tipul datei*.

Tipul datei reprezintă apartenența datei la o anumită clasă de date, căreia îi corespunde un anumit model de reprezentare internă.



Clasificarea datelor

1. **în funcție de momentul în care se produc în *fluxul de date*:**
 - date de intrare
 - date de ieșire
 - date de manevră
2. **în funcție de *valoare*:**
 - variabile
 - constante
3. **în funcție de *modul de compunere*:**
 - date elementare
 - structuri de date
4. **în funcție de *tip*:**
 - date numerice
 - date logice
 - date șiruri de caractere



Expresii

O *expresie* este constituită dintr-o succesiune de operanzi, conectați prin operatori.

Expresia este o combinație validă de operatori și operanzi.

Un *operand* poate fi o constantă, o variabilă, sau o expresie încadrată între paranteze rotunde.

Operatorii desemnează operațiile care se execută asupra operanzilor.

Forma unei expresii:

nume ← expresie

unde '←' reprezintă operator de atribuire.

În timpul execuției, expresiile sunt evaluate (adică se calculează un rezultat).



Exemple

$$E \leftarrow 8 + 12$$

$$E \leftarrow 2 * 3 + 4 / 2$$

$$E \leftarrow 3 \leq 5 \text{ or } 7 > 8$$



Operații asupra datelor

Operatori

Operatorii sunt caractere speciale (*, /, <, = etc.) sau cuvinte cheie (*mod*, *and* etc.) prin intermediul cărora se reprezintă operațiile care se efectuează în cadrul unui algoritm.

Tipuri de operatori:

- operatori *aritmetici*;
- operatori *relaționali*;
- operatori *logici*.



a. **operatori aritmetici** (matematici)

- definesc o operație aritmetică
- pot fi:
 - **unari** (se aplică asupra unui singur operand) sau
 - **binari** (acționează asupra a doi operanzi)

Operatorii aritmetici sunt de două tipuri:

- **multiplicativi**:
 - * (înmulțire)
 - / (împărțire, DIV)
 - % (restul împărțirii întregi, MOD)
- **aditivi**:
 - + (adunare)
 - (scădere)



b. operatori relaționali (de comparație)

- sunt operatori binari
- se aplică asupra operanzilor de tip numeric sau de tip șir de caractere
- furnizează un rezultat de tip logic.

Operatorii relaționali sunt: = (egal)

≠ (diferit)

< (mai mic)

> (mai mare)

≤ (mai mic sau egal)

≥ (mai mare sau egal)



c. operatori logici

- definesc o operație logică
- se pot aplica numai asupra operanzilor logici
- produc un rezultat de tip logic

Operatorii logici sunt: *not* - ! (negație logică)

and - && (*ȘI* logic)

or - || (*SAU* logic)

Notății: 1 – adevărat (true)

0 – fals (false)

x	y	! x	x y	x && y
0	0	1	0	0
0	1	1	1	0
1	0	0	1	0
1	1	0	1	1



 **Exemplu**

Se citește de la tastatură un număr natural n . Să se calculeze și să se afișeze pe ecran factorialul numărului natural n .

$$n! = 1 \cdot 2 \cdot 3 \cdot \dots \cdot n$$

Date de intrare: n

Date de ieșire: p

Date de manevră: i

unde:

n – numărul natural pentru care se calculează factorialul

i – variabilă care va memora pe rând primele n nume naturale,
 $i \in [1, n]$

p – variabilă care va memora rezultatele parțiale și rezultatul final
(produsul primelor n numere naturale nenule)



Trebuie să rezolvăm următoarea problemă:

- citirea lui n
- inițializarea variabilei p cu valoarea 1 (elementul neutru pentru produs)
- înmulțirea pe rând a numerelor $1, 2, 3, \dots, n$ la variabila p (pentru aceasta considerăm variabila i , care va lua pe rând valorile $1, 2, 3, \dots, n$, inițial valoarea lui i este 1)



Algoritmul

- Pasul 1. citește valoarea lui n ;
- Pasul 2. atribuie lui p valoarea 1;
- Pasul 3. atribuie lui i valoarea 1;
- Pasul 4. atribuie lui p valoarea $p \cdot i$;
- Pasul 5. atribuie lui i valoarea $i+1$;
- Pasul 6. dacă $i \leq n$ atunci treci la Pasul 4, altfel treci la Pasul 7;
- Pasul 7. scrie valoarea lui p .



Reprezentarea algoritmilor

Limbajul natural nu permite o descriere suficient de exactă a algoritmilor. Din acest motiv pentru reprezentarea algoritmilor se folosesc diferite forme de descriere caracteristice.

Două din cele mai folosite forme de descriere a algoritmilor sunt:

- limbajul pseudocod;
- scheme logice.



A. Reprezentarea algoritmilor în limbaj pseudocod

Limbajul pseudocod folosește *cuvinte cheie*, adică niște cuvinte cu înțeles prestabilit, care indică operația care se executată.



Exemplu

Scrieți un algoritm care calculează și afișează pe ecran suma a două numere întregi **a** și **b** citite de la tastatură.

Date de intrare: a , b

Date de ieșire: S



a). *algoritmul*

Pasul 1. solicită valori pentru **a** și **b**;

Pasul 2. calculează $S=a+b$;

Pasul 3. furnizează rezultatul pentru **s**.



b). pseudocodul

citește a,b (numere întregi)

$S \leftarrow a+b$

scrie S

În acest pseudocod s-au folosit două cuvinte cheie: *citește* și *scrie*.



Exercițiu

Fiind date trei numere naturale x , y și z , să se determine și să se afișeze pe ecran media aritmetică a celor trei numere. Valorile pentru x , y și z se citesc de la tastatură.



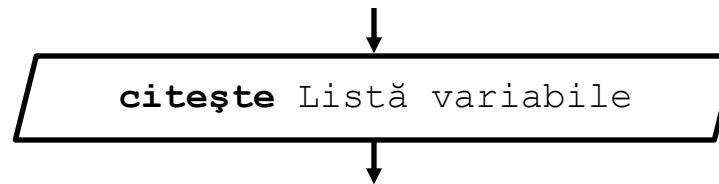
B. Reprezentarea algoritmilor prin scheme logice

Schemele logice utilizează săgeți de legătură între diferite *forme geometrice* care simbolizează acțiunile ce urmează a fi executate.

În continuare sunt prezentate blocurile care intră în componența unei scheme logice.



a. Bloc pentru introducerea datelor (bloc de citire)



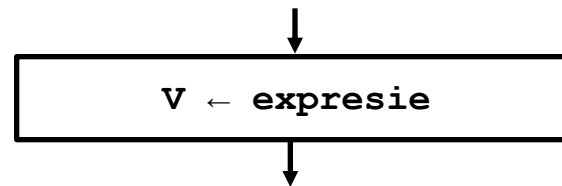
unde “**Listă variabile**” cuprinde numele simbolice ale variabilelor cărora li se asociază valori numerice (citite).

b. Bloc de extragere a rezultatelor (bloc de scriere)



unde variabilele menționate în „**Listă variabile**” constituie rezultate ale problemei.

c. Bloc de calcul (bloc de atribuire)

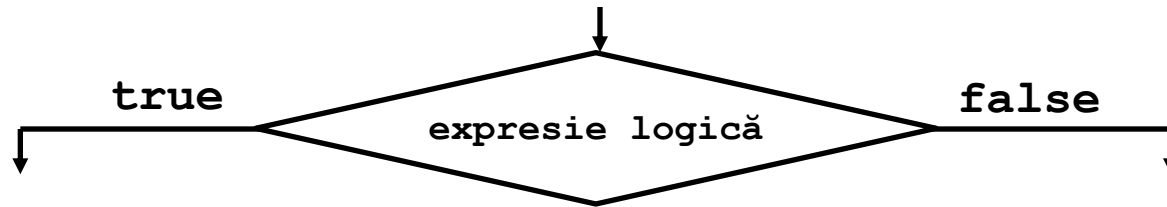


Un astfel de bloc indică următoarea succesiune de operații:

Pasul 1. se calculează expresia din membrul drept;

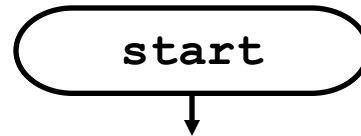
Pasul 2. se atribuie variabilei din membrul stâng valoarea calculată anterior (v reprezintă numele variabilei).

d. Bloc de decizie (bloc decizional)



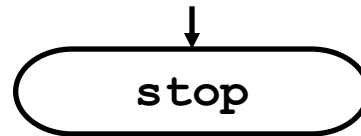
Expresie logică (condiția) poate avea valoarea “**adevărat**” sau “**fals**”. În funcție de valoarea logică obținută, blocul următor care va fi parcurs va fi legat fie de ramura “**true**”(adevărat) fie de ramura “**false**”(fals).

e. Bloc de început (bloc de start)



Indică începutul algoritmului.

f. Bloc de sfârșit (bloc de stop)



Indică sfârșitul algoritmului.

Exemplu

Scrieți un algoritm care calculează și afișează pe ecran suma a două numere naturale **a** și **b**, citite de la tastatură.

Date de intrare: a , b

Date de ieșire: S



a). *algoritmul*

Pasul 1. solicită valori pentru **a** și **b**;

Pasul 2. calculează **S=a+b**;

Pasul 3. furnizează rezultatul pentru **S**.



b). pseudocodul

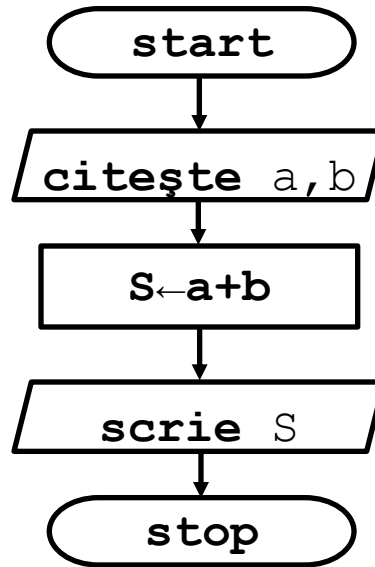
citește a,b (numere naturale)

$S \leftarrow a+b$

scrie S



c). schema logică



Exercițiu

Fiind date trei numere naturale x , y și z , citite de la tastatură, să se determine și să se afișeze media aritmetică a celor trei numere.



Principiile programării structurate

Prin structură se înțelege o anumită formă de îmbinare a operațiilor cu care lucrează algoritmi.

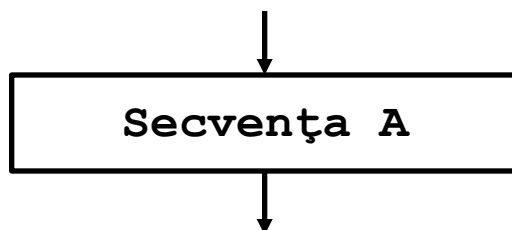
Programarea structurată are la bază o justificare matematică, furnizată de Böhm și Jacopini și cunoscută ca “teorema de structură” care precizează că orice algoritm având o *intrare* și o *ieșire* (adică un singur punct de început și un singur punct de terminare a execuției) poate fi reprezentat ca o combinație a trei structuri de control:

- a. **secvența** – succesiune de două sau mai multe operații;
- b. **decizia** – alegerea unei operații din două alternative posibile;
- c. **repetiția** – repetarea unei operații atâta timp cât o anumită condiție este îndeplinită.



a. Structura secvențială (liniară)

Secvența reprezintă o succesiune de două sau mai multe operații care conține o transformare de date:



unde “Secvența A” reprezintă o transformare de date.

În limbaj natural, execuția poate fi descrisă astfel: pașii se execută în ordinea în care au fost scriși.

În pseudocod, execuția se descrie astfel:

`operație 1`

`operație 2`

`.....`

`operație n`



Exemplu

Să se calculeze și să se afișeze pe ecran suma, produsul și diferența a trei nume întregi x , y și z citite de la tastatură.

Date de intrare: x , y , z

Date de ieșire: S , P , D



a). *algoritmul*

Pasul 1. se dau valori pentru x , y și z ;

Pasul 2. se calculează $S=x+y+z$

Pasul 3. se calculează $P=x*y*z$

Pasul 4. se calculează $D=x-y-z$

Pasul 5. afișează rezultatele pentru S , P și D .



b). pseudocodul

citește x, y, z (numere întregi)

$S \leftarrow x + y + z$

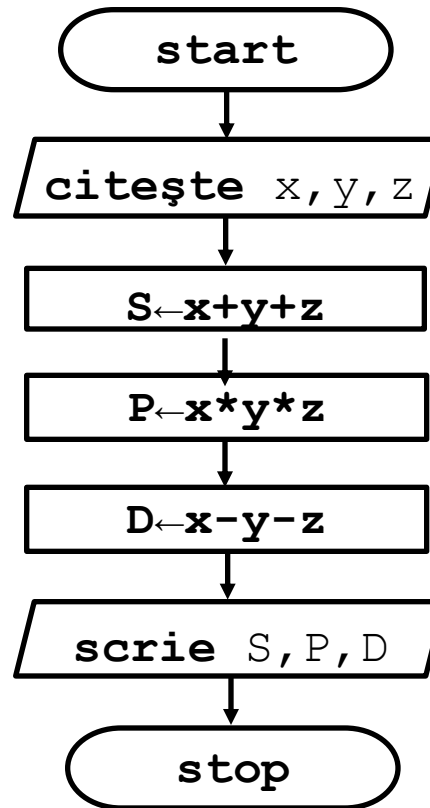
$P \leftarrow x * y * z$

$D \leftarrow x - y - z$

scrie S, P, D



c). schema logică



Exercițiu

Se dau trei numere naturale **a**, **b** și **c** citite de la tastatură. Să se calculeze și să se afișeze pe ecran valorile expresiilor:

$$S1 = (a+b) \cdot (a-b)$$

$$S2 = a \cdot b + a \cdot c + b \cdot c$$

$$P = S1 \cdot S2$$

Se cer:

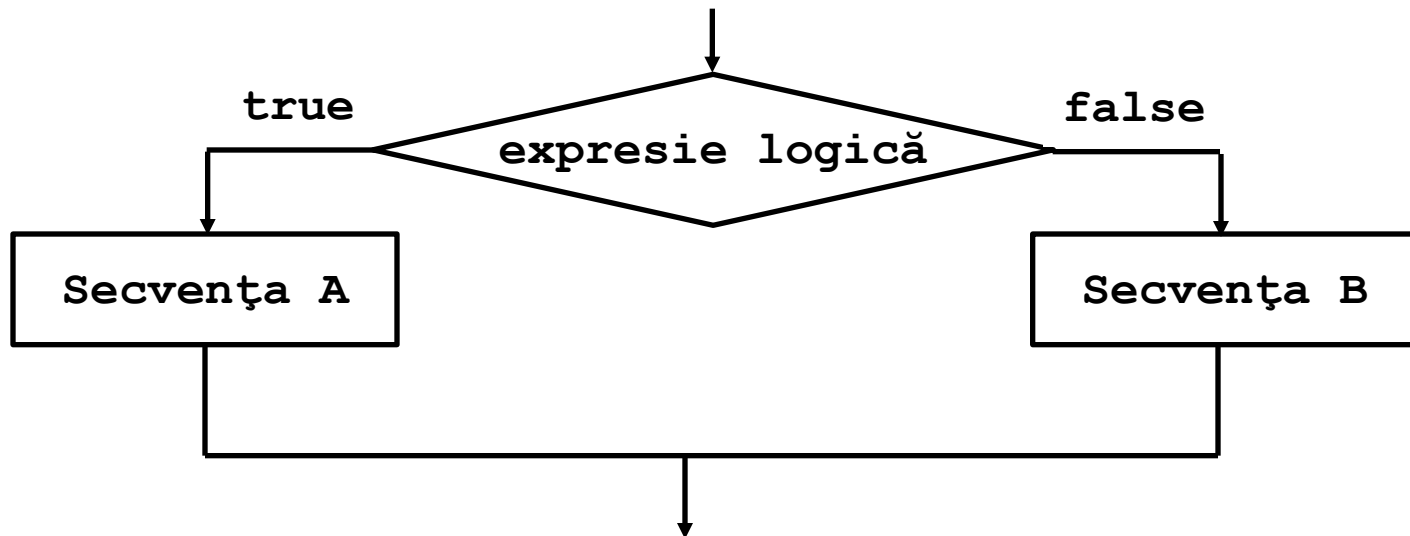
- a) algoritmul;
- b) pseudocodul;
- c) schema logică.



b. Structura decizională (alternativă)

Decizia reprezintă alegerea unei operații sau a unei secvențe de operații dintre două alternative posibile.

Structura decizională are următoarea formă:



În limbaj natural, execuția poate fi descrisă astfel:

Pasul 1. se evaluează expresia logică;

Pasul 2. dacă aceasta este adevărat, se execută “Secvența A”;

în caz contrar (dacă este falsă) se execută “Secvența B”.



În pseudocod, execuția se descrie astfel:

```
dacă expresie logică atunci  
    Secvența A  
altfel  
    Secvența B
```



Exemplu

Se dau două numere naturale **a** și **b** distincte. Să se determine care dintre ele are valoarea mai mare.

Date de intrare: a , b

Date de ieșire: max



a). *algoritmul*

Pasul 1. se dau valori lui **a** și **b**;

Pasul 2. se determină maximul dintre **a** și **b**: dacă **a** este mai mare ca **b** atunci maximul este **a** altfel maximul este **b**;

Pasul 3. se afișează maximul.

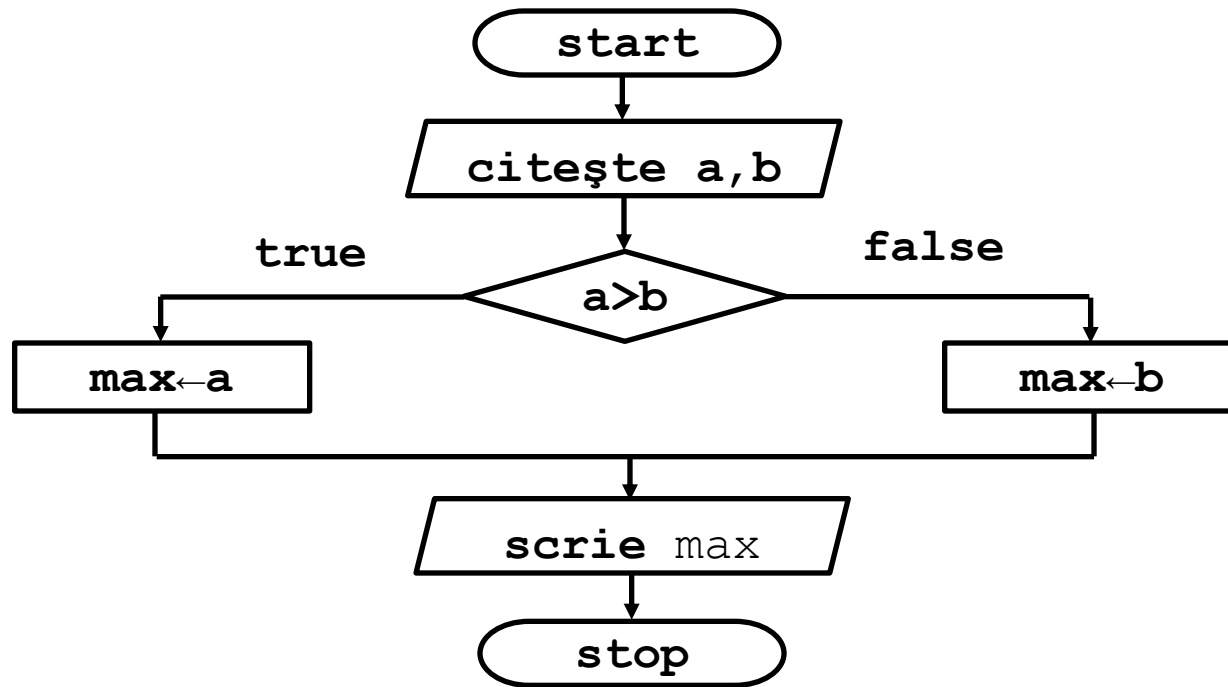


b). pseudocodul

```
citește a,b (numere naturale)  
  dacă a>b atunci  
    max←a  
  altfel  
    max←b  
  ■  
scrie max
```



c). schema logică



Exercițiu

Se citește de la tastatură un număr întreg x . Să se afișeze mesajul “Da” dacă numărul citit este pozitiv sau mesajul “Nu” în caz contrar.

Se cer:

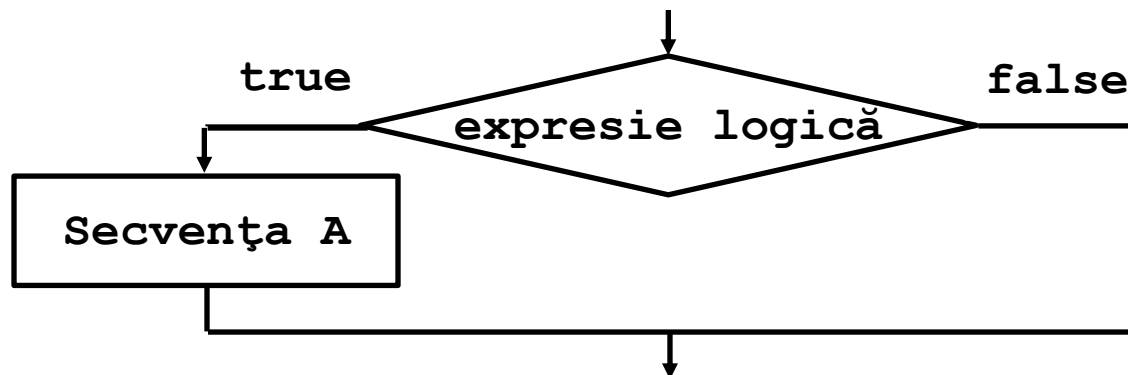
- a) algoritmul;
- b) pseudocodul;
- c) schema logică.



Decizia cu varianta unei căi nule

Mai există o formă a structurii decizionale și anume cea cu varianta unei căi nule.

Forma acestei structuri este următoarea:



În limbaj natural, execuția poate fi descrisă astfel:

Pasul 1. se evaluează expresia logică;

Pasul 2. dacă aceasta este adevărat, se execută “**Secvența A**” apoi execuția structurii decizionale se încheie; în caz contrar (dacă expresia logică este falsă) execuția structurii decizionale se încheie.



În pseudocod, execuția se descrie astfel:

```
dacă expresie logică atunci  
    Secvența A  
■
```



Exemplu

Se citește de la tastatură o valoare întreagă **a**. Să se afișeze pe ecran modul numărului **a**.

Date de intrare: a

Date de ieșire: a



a). *algoritmul*

Pasul 1. se dă valoare lui a ;

Pasul 2. dacă a este negativ atunci a devine pozitiv;

Pasul 3. se tipărește a .



b). pseudocodul

citește a (număr întreg)

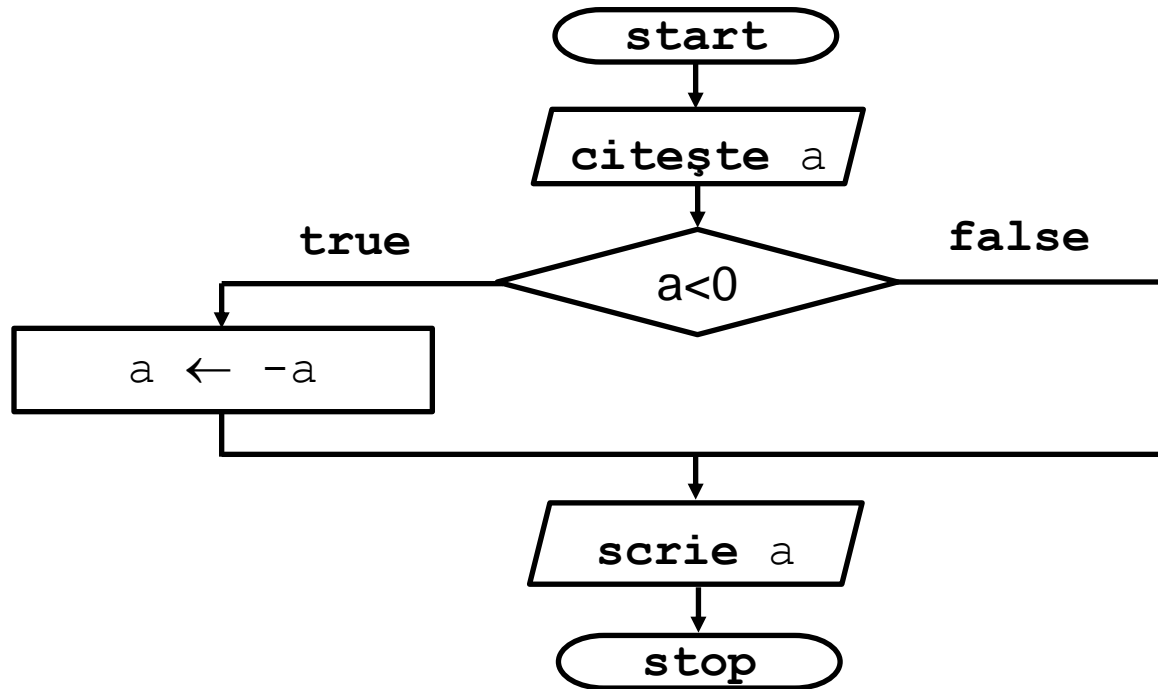
dacă $a < 0$ **atunci**

$a \leftarrow -a$
■

scrie a



c). schema logică



Exercițiu

Se citește de la tastatură un număr întreg x . În cazul în care acesta este mai mare sau egal cu zero, se va afișa pe ecran mesajul “**număr natural**”, altfel nu se va afișa niciun mesaj.

Se cer:

- a) algoritmul;
- b) pseudocodul;
- c) schema logică.



c. Structura repetitivă

Repetiția (bucla, ciclul sau iterația) asigură execuția unei secvențe în mod repetat în funcție de o anumită condiție.

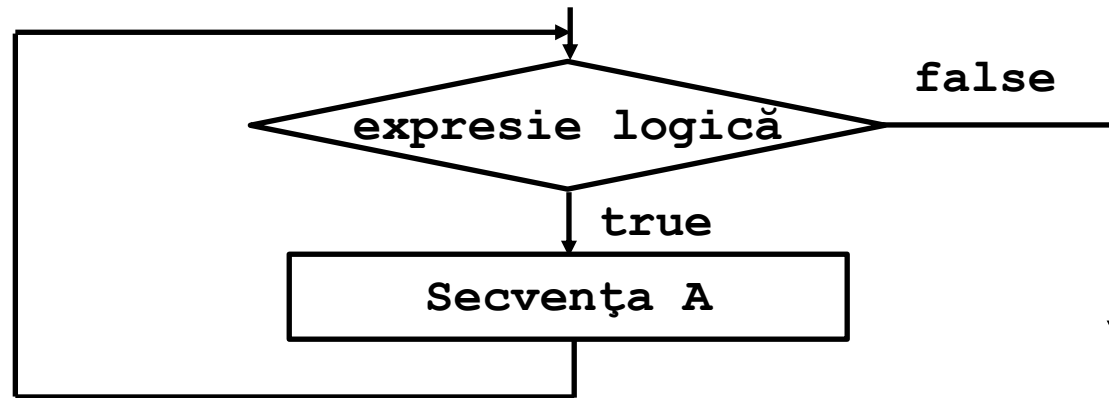
Există trei tipuri de structuri repetitive:

- structura repetitivă *cu test inițial*;
- structura repetitivă *cu test final*;
- structura repetitivă *cu contor*.



a). Structura repetitivă cu test inițial

Structura repetitivă cu **test inițial** are forma:



Execuția structurii repetitive cu test inițial presupune parcurgerea următoarelor etape:

Pasul 1. se evaluează expresia logică; dacă rezultatul este adevărat se trece la Pasul 2, altfel execuția se încheie;

Pasul 2. se execută „**Secvența A**”, apoi se trece la Pasul 1.



Exprimarea în pseudocod:

```
cât timp expresie logică execută  
  Secvența A
```



Exemplu

Să se calculeze și să se afișeze pe ecran suma primelor n numere naturale nenule. Valoarea lui n (număr natural nenul) se citește de la tastatură.

Date de intrare: n

Date de ieșire: S

Date de manevră: i



a). algoritmul

Pasul 1. se citește valoarea lui n ;

Pasul 2. se atribuie lui S valoarea 0 și lui i valoarea 1;

Pasul 3. dacă i este mai mic sau egal cu n se trece la Pasul 4, altfel se trece la Pasul 5;

Pasul 4. se calculează suma după formula $S=S+i$ și i ia valoarea următorului termen al sumei, după formula $i=i+1$, apoi se trece la Pasul 3;

Pasul 5. se afișează valoarea sumei S .



b). pseudocodul

citește n (număr natural nenul)

S←0

i←1

cât timp $i \leq n$ **execută**

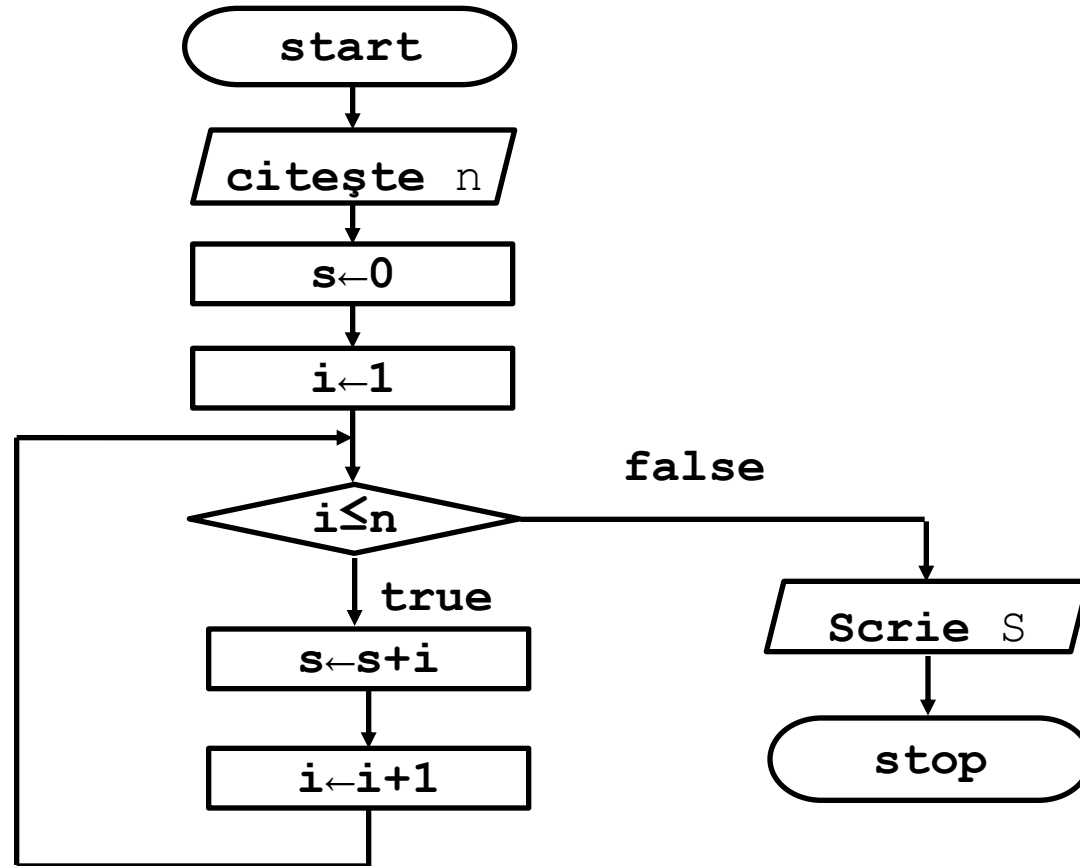
S←S+i

i←i+1



scrie S

c). schema logică



Exercițiu

Să se calculeze și să se afișeze pe ecran produsul primelor n numere naturale nenule. Valoarea lui n (număr natural nenul) se citește de la tastatură.

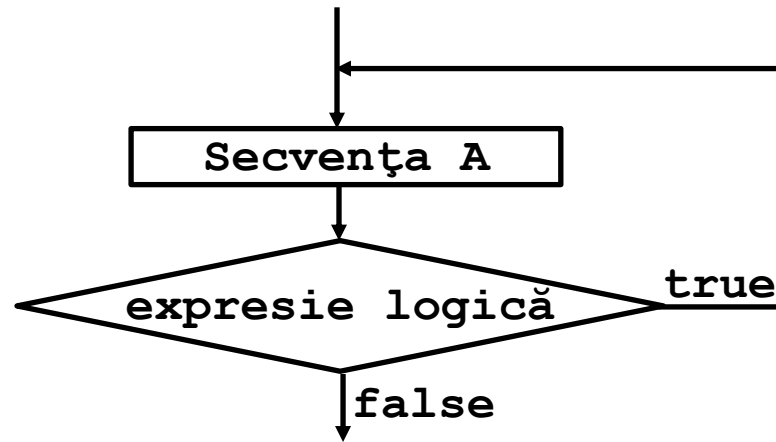
Se cer:

- a) algoritmul;
- b) pseudocodul;
- c) schema logică.



b. Structura repetitivă cu test final

Structura repetitivă cu **test final** are forma:



Execuția structurii repetitive cu test final presupune parcurgerea următoarelor etape:

Pasul 1. se execută secvența A

Pasul 2. se evaluează expresia logică; dacă rezultatul este adevărat, se continuă cu pasul 1, în caz contrar, se încheie execuția structurii repetitive.



Exprimarea în pseudocod:

```
execută  
    Secvența A  
cât timp expresie logică
```



Exemplu

Să se calculeze și să se afișeze pe ecran suma primelor n numere naturale nenule. Valoarea lui n (număr natural nenul) se citește de la tastatură.

Date de intrare: n

Date de ieșire: S

Date de manevră: i



a). algoritmul

Pasul 1. se citește valoarea lui n ;

Pasul 2. se atribuie lui S valoarea 0 și lui i valoarea 1;

Pasul 3. se calculează suma $S=S+i$ și i ia valoarea $i=i+1$;

Pasul 4. dacă $i \leq n$ se trece la Pasul 3, în caz contrar se trece la Pasul 5;

Pasul 5. se afișează valoarea sumei S .



b). pseudocodul

citește n (număr natural nenul)

S←0

i←1

┌ **execută**

 S←S+i

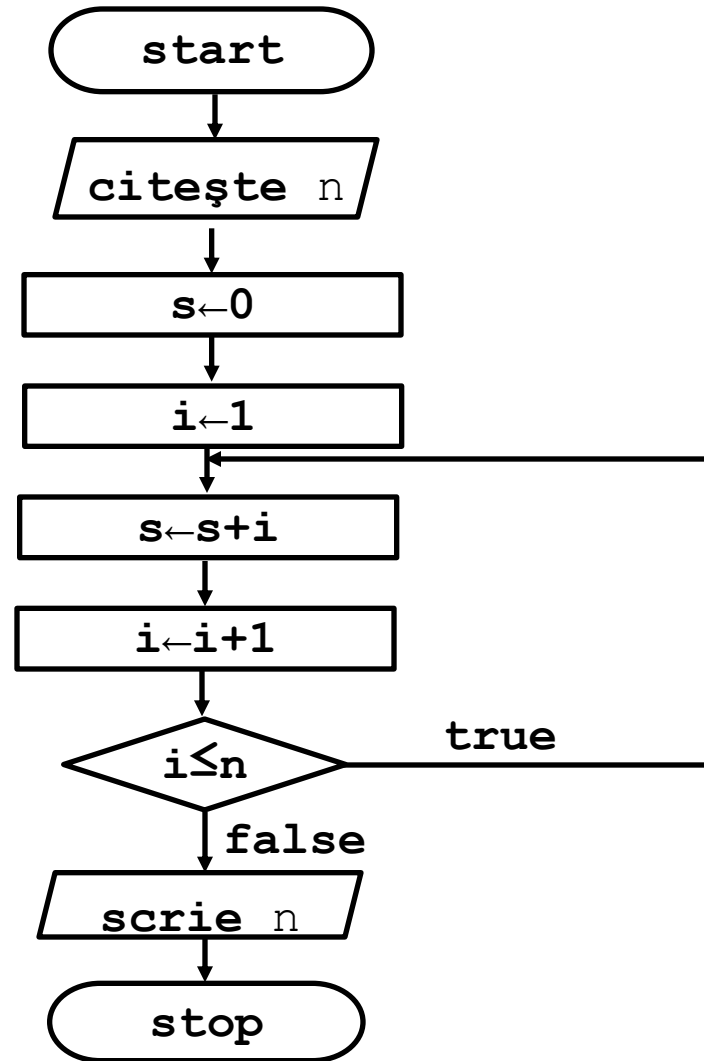
 i←i+1

└ **cât timp** $i \leq n$

scrie S



c). schema logică



Exercițiu

Să se calculeze și să se afișeze pe ecran produsul primelor n numere naturale nenule. Valoarea lui n (număr natural nenul) se citește de la tastatură.

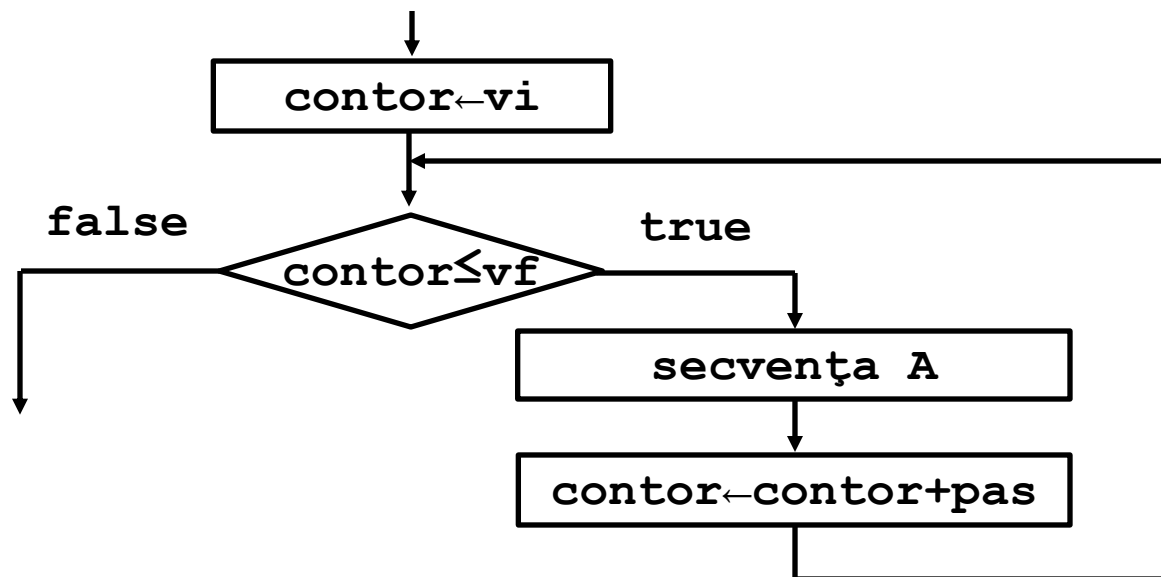
Se cer:

- a). algoritmul;
- b). pseudocodul;
- c). schema logică.



c. Structura repetitivă cu contor

Structura repetitivă cu **contor** are forma:



unde cu “**vi**” s-a notat valoarea inițială, iar cu “**vf**” s-a notat valoarea finală.

Execuția structurii repetitive cu contor presupune parcurgerea următoarelor etape:

Pasul 1. variabila de ciclare “**contor**” ia valoarea inițială “**vi**”;

Pasul 2. dacă “**contor**” este mai mic sau egal cu valoarea finală “**vf**”, se execută “**Secvența A**”, se adună valoarea pasului la “**contor**” și se reia cu pasul 2, altfel, execuția este încheiată.



Exprimarea în pseudocod:

```
pentru contor=vi, vf, pas execută  
    secvența A
```

Observație: dacă valoarea pasului este 1, nu este obligatorie specificarea pasului.



Exemplu

Să se calculeze și să se afișeze pe ecran suma primelor n numere naturale. Valoarea lui n (număr natural nenul) se citește de la tastatură.

Date de intrare: n

Date de ieșire: S

Date de manevră: i



a). algoritmul

Pasul 1. se dă valoare lui n ;

Pasul 2. se dă lui S valoarea 0 și lui i valoarea 1 ;

Pasul 3. dacă $i \leq n$ se trece la Pasul 4, altfel se trece la Pasul 5 ;

Pasul 4. se calculează suma după formula $S=S+i$ și $i=i+1$;

Pasul 5. se afișează valoarea sumei S .



b). pseudocodul

citește n (număr natural nenul)

S ← 0

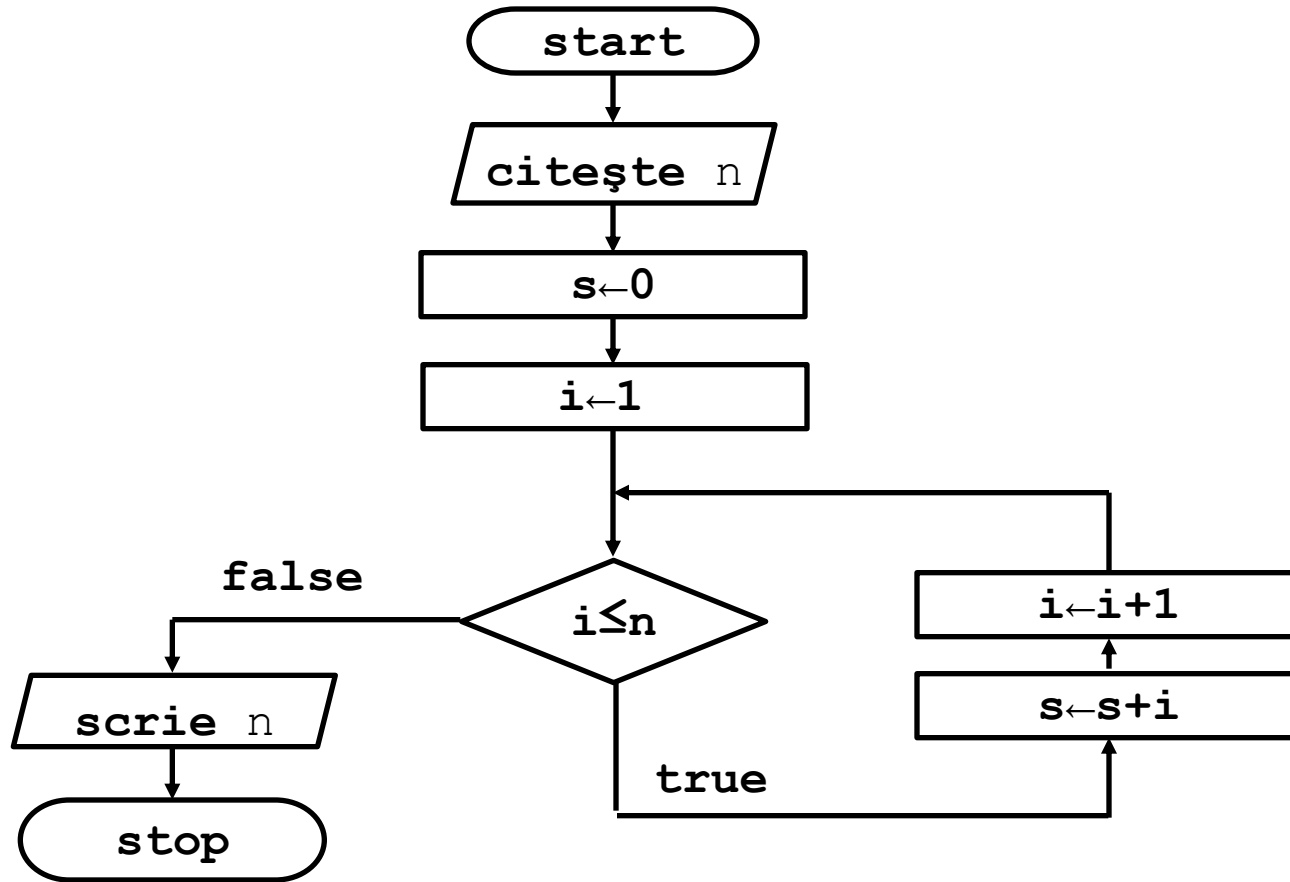
pentru i=1, n **execută**

 S ← S + i

scrie S



c). schema logică



Exercițiu

Să se calculeze și să se afișeze pe ecran produsul primelor n numere naturale. Valoarea lui n (număr natural nenul) se citește de la tastatură.

Se cer:

- a). algoritmul;
- b). pseudocodul;
- c). schema logică.



Analiza eficienței unui algoritm

Pentru rezolvarea unei probleme se pot folosi mai mulți algoritmi. În acest caz se va alege algoritmul cel mai eficient.

Algoritmul cel mai eficient este cel care folosește cel mai puțin resursele calculatorului și anume:

- **memoria internă**;
- **procesorul**.



A. Memoria internă

Pentru a face economie de memorie internă, trebuie avute în vedere următoarele:

- alegerea corectă a tipului de dată pentru fiecare variabilă de memorie folosită;
- rezolvarea problemei folosind cât mai puține variabile de memorie.

Analiza datei trebuie să se facă în două moduri la alegerea tipului de dată:

- logic;
- fizic.



a. Logic (la nivel conceptual)

- analiza se face pornind de la enunțul problemei;
- presupune identificarea *domeniului de definiție extern* al datelor.

Exemplu de domeniu de definiție extern al datelor: pentru a memora un număr de la extragerea loto “6 din 49” se va folosi o variabilă care memorează un număr întreg pozitiv din intervalul $[1,49]$.



- b. Fizic (la nivelul reprezentării ei în memoria internă)
- analiza se face pornind de la tipurile de date implementate în limbajul de programare;
 - fiecare tip de dată are un *domeniu de definiție intern* al datelor;
 - se alege tipul de dată care consumă cea mai puțină memorie, astfel încât domeniul de definiție extern al datei să fie inclus în domeniul de definiție intern al datei.



B. Procesorul

Timplu de execuție al unui algoritm depinde de câte valori ale datelor de intrare vor fi prelucrate.

Se definește *dimensiunea datelor de intrare* ca fiind numărul de valori pentru datele de intrare ale unui algoritm.

În funcție de complexitatea algoritmului, evaluarea timpului de execuție se poate face prin:

- numărul de operații elementare ale algoritmului, sau
- timpul mediu al programului.



- a. Numărul de operații elementare ale algoritmului
- o operație elementară este o operație sau o succesiune de operații care nu depind de caracteristicile problemei (exp: o operație de citire/scriere, o operație aritmetică, o operație de comparație, o operație de atribuire, un grup de astfel de operații);
 - în schimb, n operații de atribuire nu reprezintă o operație elementară, deoarece depinde de o caracteristică a problemei, și anume de valoarea lui n ;



Exemplu

Suma a n numere

```
citește n (număr natural)
s←0
pentru i←1,n execută
    citește a
    s ← s+a
scrie s
```

Suma a n numere pare

```
citește n (număr natural)
s←0
pentru i←1,n execută
    citește a
    dacă a mod 2=0 atunci
        s ← s+a
scrie s
```

Numărul de operații:

<i>Suma a n numere</i>	<i>Suma a n numere pare</i>
<ul style="list-style-type: none">- 4 operații elementare în afara structurii repetitive: citire, atribuire, inițializare contor, scriere- 4 operații elementare în structura repetitivă: comparare contor, incrementare contor, citire, atribuire	<ul style="list-style-type: none">- 4 operații elementare în afara structurii repetitive: citire, atribuire, inițializare contor, scriere- 4 operații elementare în structura repetitivă: comparare contor, incrementare contor, citire, Comparare
Total operații: $4+4*n$	<ul style="list-style-type: none">- nu se poate preciza dacă se execută și atribuirea (x reprezintă numărul elementelor pare)
	Total operații: $4+4*n+x$



b. Timpul mediu al programului

- *timpul mediu de execuție* depinde de *timpul minim de execuție* care corespunde cazului cel mai favorabil (când se execută cele mai puține operații) și *timpul maxim de execuție* care corespunde cazului cel mai defavorabil (când se execută cele mai multe operații);



Exemplu

Suma a n numere pare.

- cazul cel mai favorabil: nu există numere pare ($x=0$) și numărul de operații este $4+4*n$;

- cazul cel mai defavorabil: toate numerele sunt pare ($x=n$) și numărul de operații este $4+4*n+n \leftrightarrow 4+5*n$;

- pentru a calcula timpul mediu de execuție, va trebui să analizăm toate cazurile posibile (în total $n+1$ cazuri): niciun număr par, un număr par, două numere pare, ..., n numere pare;

- numărul mediu de operații fiind dat de x calculat astfel: $\frac{0+1+2+\dots+n}{n+1} = \frac{\frac{n \times (n+1)}{2}}{n+1} = \frac{n}{2}$



Fișă de lucru

- Algoritmi



10. Bibliografie & webografie

1. Miloşescu M., *Informatică. Manual pentru clasa a IX-a*, Editura Didactică și Pedagogică, București, 2004
2. Munteanu F., *Programarea calculatoarelor. Manual pentru licee de informatică clasele X-XII*, Editura Didactică și Pedagogică, București, 1994
3. Niculescu S., Eftene S., *Algoritmi și limbaje de programare*, Editura Didactică și Pedagogică, București, 1995
4. Cerchez E., Şerban M., *Informatică. Manual pentru clasa a IX-a*, Editura Didactică și Pedagogică, București, 2004
5. Sorin T., *Informatică. Manual pentru clasa a IX-a*, Editura L&S Infomat, 2000
6. Popescu C., *Culegere de probleme de informatică*, Editura Donaris, Sibiu, 2002
8. Ministerul Educației, Cercetării și Tineretului, Centrul Național pentru Curriculum și Evaluare în Învățământul Preuniversitar, *Proba scrisă la informatică. Examenul de bacalaureat – Variante (1-100)*, București 2008

