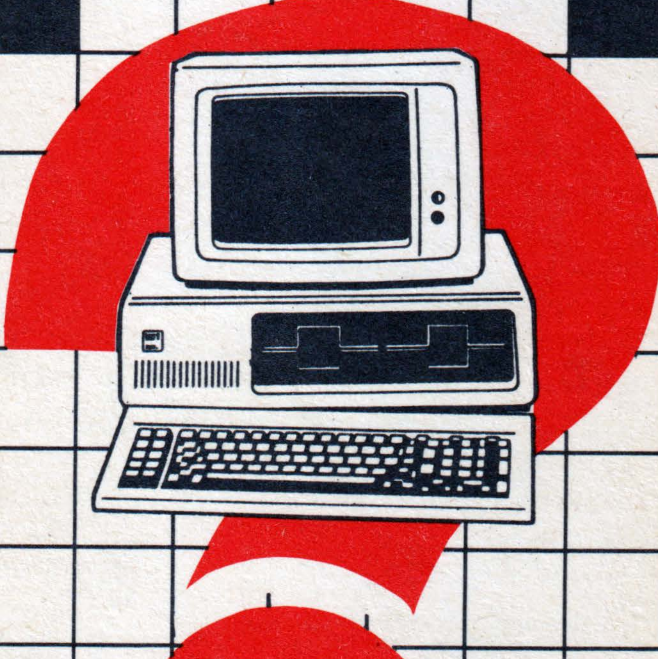
ADRIAN ATANASIU

Cum se scrie un ALGORITM SIMPLU!



Editura Agni

Biblioteca de Informatică

Adrian Atanasiu

Cum se scrie un algoritm? Simplu

Editura Agni

Bucureşti 1993

Redactor : *Victor Mitrana*
Tehnoredactare : *Vlad Atanasiu*
Aurelian Lavric
Coperta: *Adina Dumitriu*

Tiparul executat la Sucursala Poligrafică „Bucureştii Noi"

Familiei mele...

Autorul

ISBN 973-95626-5-5

Editura AGNI,
CP:30-107, BUCUREŞTI

Cărţile Editurii AGNI se pot cumpăra şi la magazinul DOMINI,
Str. Pictor Verona 18, Bucureşti (lîngă cinematograful Patria).

Nu luaţi nimic de bun apriori
dacă puteţi verifica .
(R. Kipling)

CUVÎNT ÎNAINTE

Prin dezvoltarea sa, informatica se impune tot mai mult ca o disciplină de studiu în şcoală. Odată ce sîntem de acord cu această necesitate, apar două probleme:

a. La ce moment să fie introduse în programa şcolară ore de informatică;

b. Ce tip de conţinut să aibă aceste ore ?

Astfel, putem vorbi despre ideea de **`utilizatori de calculatoare`** conform cărora, după învăţarea mînuirii lor, elevii să " **butoneze** " diverse pachete de programe - atît cu subiect de divertisment (jocuri, efecte) cît şi pedagogic (pentru prezentarea de teme din fizică, matematică, geografie, muzică etc).

Există însă şi un conţinut creativ: învăţarea de limbaje de programare prin intermediul cărora putem conversa cu calculatorul, îl putem solicita să ne ajute la rezolvarea unor probleme pe care le avem.

Să încercăm să răspundem pe rînd la aceste chestiuni.

a. Experienţa demonstrează că întîlnirea cu calculatorul este benefică de la o vîrstă cît mai fragedă. Copiii nu au prejudecăţi, mintea lor este permanent iscoditoare în căutare de noutăţi; pentru ei totul este ca o joacă în care descoperă mereu ceva nou. Sub o îndrumare competentă, saltul este spectaculos. De aceea, cred că aici răspunsul poate fi: în funcţie de dotare şi de profesor, întîlnirea cu calculatorul trebuie să se facă cît mai devreme posibil. Ţinîndu-se însă cont de discernămîntul copiilor recomandăm ca orele de informatică cu o programă coerentă să se introducă cam prin clasa V.

b. Dacă dotarea spaţiului de învăţămînt permite acest lucru, este bine să se înceapă cu prima fază - aceea de **utilizator**. Astfel, copilul este atras imediat de calculator. În plus, permite celorlaltor discipline ca, prin reorganizarea materiei să treacă la un alt sistem de învăţare, mai atractiv şi mai eficient .

Odată cu dezvoltarea copilului, acesta refuză să fie doar o componentă pasivă, aservită calculatorului. Nevoia lui permanentă de a şti solicită trecerea la o a doua fază în care maşina nu mai este totul, ci doar o parte a unui întreg sistem informaţional.

Bineînţeles, toate cunoştinţele de informatică trebuiesc îmbinate cu celelalte materii care se studiază în paralel (în special matematică şi fizică). Elementele teoretice introduse treptat readuc calculatorul la poziţia sa de fapt, aceea de unealtă în sprijinul gîndirii.

Lucrarea de faţă a fost concepută ca un proiect de manual, fiind doar o componentă dintr-un ciclu mai larg. Astfel, în intenţia noastră, o carte de informatică de clasa V-a trebuie să conţină probleme simple de logică (sub formă de jocuri) precum şi o prezentare a calculatorului; în clasele VI-VII se pot învăţa două limbaje de programare scrise special pentru asemenea vîrstă (LOGO şi BASIC) iar pentru clasa VIII să fie prevăzut un studiu teoretic elementar despre conţinutul unui program (deci ideea de algoritm).

Acum, răsfoind din nou cartea, realizez că totul este perfectibil; acesta este doar un drum posibil. De aceea sînt în aşteptarea de propuneri şi/sau critici. Să nu uităm totuşi că scopul urmărit este cel prezentat la început anume acela că **învăţămîntul informatic trebuie să existe.**

Conf. dr. Adrian Atanasiu

Adevărul nu se află la capătul unui drum,
ci la acela care deschide o cale .
(Gelu Negrea Gheorghe)

CUPRINS

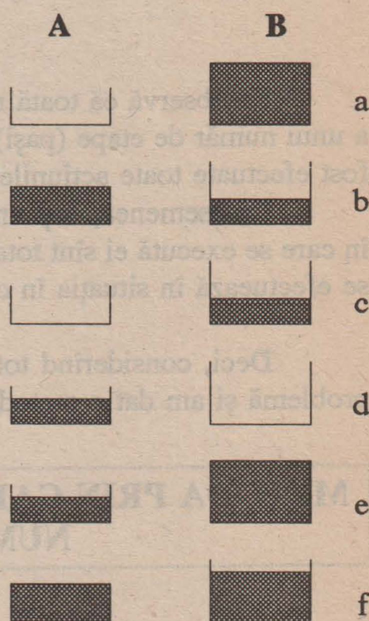
I. ALGORITMI - PREZENTARE
I.1 Introducere

Să presupunem că avem de rezolvat următoarea problemă:

Sînt două vase: unul de 3 litri şi altul de 5 litri. Cum facem să avem 4 litri de apă în vasul de 5 litri ?

Se subînţelege că nu dispunem de nici un alt mijloc de măsurare a lichidelor precum şi de faptul că putem vehicula prin cele două vase o cantitate nelimitată de apă.

Fie **A** vasul de 3 litri şi **B** - cel de 5 litri.
Soluţia este dată de următoarele acţiuni:



a. Se umple vasul **B** cu apă;

b. Cu apa din vasul **B** se umple vasul **A**;
Astfel, în **A** avem 3 l iar în **B** - 2 litri.

c. Se aruncă apa din vasul **A**;

d. Se pune tot restul de apă din **B** în **A**;
Deci **A** conţine 2 litri de apă iar **B** este gol.

e. Se umple vasul **B** cu apă;

f. Se toarnă apa din **B** pînă se umple **A** .
Pentru ca **A** să fie plin, mai trebuie turnat un litru. Dacă din cei 5 litri din **B** turnăm un litru în **A**, mai rămîn în **B** 4 litri, exact soluţia problemei.

Să rescriem soluţia, puţin mai formalizat:
Vom nota **A ←---** , **B ←---** umplerea vaselor,
←--- A , **←--- B** golirea lor,
A←-^{n}--B turnarea de n litri de apă din **B** în **A** .

Deci

a. **B ←---**

b. **A←-^{3}--B**

c. ←--- **A**

d. **A←-^{2}--B**

e. **B ←---**

f. **A←-^{1}--B**

Se observă că toată rezolvarea problemei constă în parcurgerea pe rînd a unui număr de etape (paşi). Fiecare pas nu poate fi executat decît după ce au fost efectuate toate acţiunile dinaintea lui.

De asemenea, deşi unii paşi par identici (cum ar fi (a) şi (e)), momentele în care se execută ei sînt total diferite. Astfel, la (a) vasul **A** era gol, pe cînd (e) se efectuează în situaţia în care vasul **A** are 2 litri de apă.

Deci, considerînd totul la un mod general, ce am făcut ? Am avut o problemă şi am dat o metodă de soluţionare a ei.

METODA PRIN CARE SE REZOLVĂ O PROBLEMĂ SE NUMEŞTE ALGORITM.

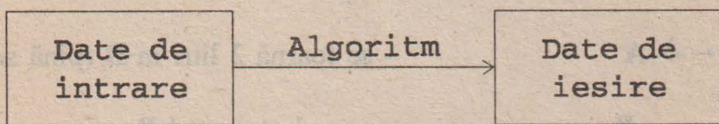
Din problema de sus se desprind două informaţii:

- rezultatul final: 4 litri de apă;
- metoda (algoritmul) prin care se ajunge la acest rezultat: paşii (a)-(e).

Cele două informaţii sînt complet deosebite.

Astfel, pentru o problemă dată, rezultatul final este unic. În marea majoritate a cazurilor, el şi numai el este acela care interesează pe cel care a pus problema (utilizator).

Putem reprezenta schematic această situaţie astfel:



Un utilizator trebuie să înveţe cum să introducă datele de intrare şi cum să citească rezultatul. "Cutia neagră"-dreptunghiul în care datele iniţiale se transformă în date finale - este ignorată. Calculatorul este un mediu ideal pentru astfel de solicitări.

La un calculator de buzunar de exemplu, nu facem altceva decît să "butonăm" numerele şi operaţiile care trebuiesc efectuate; calculatorul afişează rezultatul pe ecran. Cum ajunge la acest rezultat, nu ştim. Ceea ce vrem este ca el să fie obţinut

- rapid
- corect

Acea cutie neagră devine însă interesantă în două situaţii:

- Cînd nu există la dispoziţie un algoritm pentru rezolvarea problemei (şi deci acesta trebuie creat);
- Un astfel de algoritm există dar, din diverse motive el este inabordabil pentru utilizator (este prea scump, este prea greoi, este greşit, nu răspunde la datele de intrare solicitate etc.).

În general, pentru o problemă dată, dacă există un algoritm de rezolvare, acesta nu este unic.

De exemplu, la problema anterioară se poate da şi o altă metodă de rezolvare:

a. **A** ←--- - se umple vasul **A** (de 3 litri)

b. $\mathbf{B} \xleftarrow{3} \mathbf{A}$ - se toarnă 3 litri în **B**

c. **A** ←--- - se umple din nou vasul **A**

d. $\mathbf{B} \xleftarrow{2} \mathbf{A}$ - se toarnă 2 litri în **B** (pînă se umple)

e. ←--- **B** - se goleşte vasul **B**

f. $\mathbf{B} \xleftarrow{1} \mathbf{A}$ - se goleşte vasul **A** în **B** (cu un litru)

g. **A** ←--- - vasul **A** se umple din nou

h. $\mathbf{B} \xleftarrow{3} \mathbf{A}$ - cei 3 litri din **A** se toarnă în **B** (care are acum 4 litri).

Cînd se pot imagina mai mulţi algoritmi diferiţi pentru rezolvarea unei probleme, ne putem întreba pe ce criterii să alegem unul mai bun.

"**Mai bun** " în ce sens ?

Putem spune că un algoritm este "mai bun" dacă este mai rapid (de exemplu al doilea algoritm are 8 paşi, pe cînd primul - doar 6), conţine mai puţine operaţii şi locaţii, prevede mai multe situaţii la care să răspundă, etc.

Să mai studiem o problemă:

Pe malul unei ape se află un om împreună cu un lup, o capră şi o varză. El trebuie să treacă apa cu tot ce îi aparţine; are la dispoziţie o singură barcă care poate duce la o traversare doar pe om însoţit de un singur element din celelalte. Dar, dacă ar rămîne nesupravegheate pe un mal, lupul ar mînca capra, iar capra - varza. Cum trebuie să procedeze omul ?

Algoritmul, foarte cunoscut, poate fi prezentat în diverse forme. Folosind puţină formalizare însă, putem obţine o prezentare mai clară a problemei şi o familie infinită de algoritmi pentru rezolvarea ei.

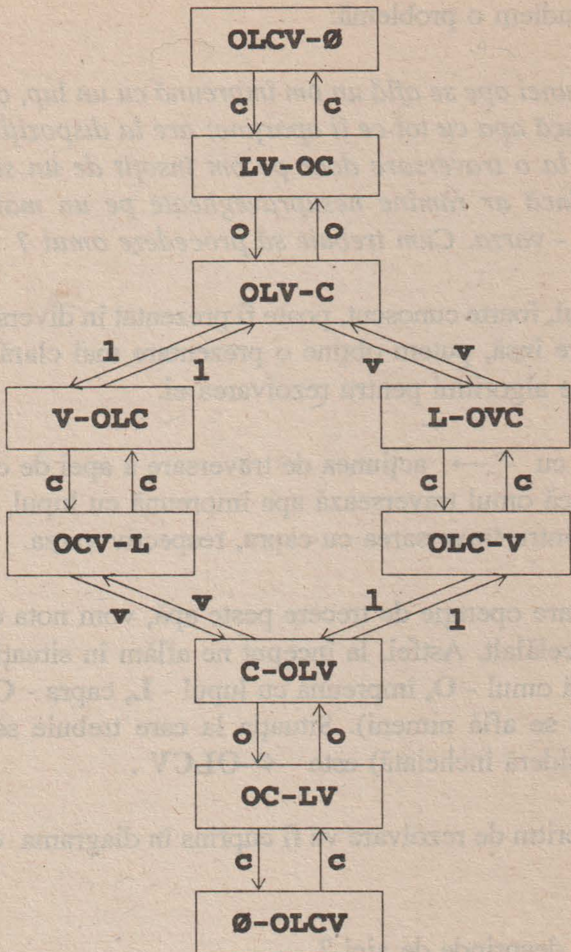
Să notăm cu $\xrightarrow{o}$ acţiunea de traversare a apei de către om singur în barcă $\xrightarrow{l}$ dacă omul traversează apa împreună cu lupul, $\xrightarrow{c}$, $\xrightarrow{v}$ similar pentru traversarea cu capra, respectiv varza.

După fiecare operaţie de trecere peste apă, vom nota cine se află pe un mal şi cine - pe celălalt. Astfel, la început ne aflăm în situaţia **OLCV-∅** (pe un mal se află omul - **O**, împreună cu lupul - **L**, capra - **C** şi varza - **V** ; pe malul celălalt nu se află nimeni). Situaţia la care trebuie să se ajungă (cînd problema se consideră încheiată) este **∅-OLCV** .

Orice algoritm de rezolvare va fi cuprins în diagrama descrisă în pagina următoare.

Ce putem desprinde de aici ?

Că problema nu are o singură soluţie, ci o infinitate. Fiecare drum care pleacă de sus şi ajunge în punctul de jos constituie un algoritm. Astfel omul se poate plimba de pe un mal pe altul de cîte ori vrea, ducînd sau nu şi pe altcineva în barcă, fără ca asta să genereze o soluţie greşită (de exemplu, un algoritm ar putea fi - notînd numai acţiunile de traversare: `covclvclvcloc`).



Problema, pentru a fi completă, ar trebui să specifice din cîte traversări să fie rezolvată. Dacă s-ar cere numărul minim de traversări, am avea doi algoritmi: `covcloc` şi `colcvoc` (şi numărul minim de traversări ar fi 7).

Altfel, pentru orice număr impar de traversări mai mare decît 7, numărul de algoritmi posibili creşte mult.

Deci un algoritm trebuie să completeze în construcţia sa anumite lacune (lăsate intenţionat sau nu) din enunţul problemei.

NU EXISTĂ UN ALGORITM DE SCRIERE A ALGORITMILOR.

Reprezentarea unui algoritm este determinată de două componente:

✧ **generală**- un algoritm se poate scrie într-o modalitate cunoscută de toţi cei care au nevoie de acest algoritm. Această modalitate intermediară, pe care o pot folosi doar anumite categorii de utilizatori, este aceea de pseudocod sau de schemă logică.

✧ **aplicativă** - cînd se ştie suportul exact pe care va opera algoritmul. Această reprezentare este limbajul de programare.

BASIC este o modalitate aplicativă de reprezentare (folosit numai cînd se lucrează pe un calculator care "cunoaşte" limbajul BASIC - sau o anumită variantă de limbaj BASIC; Q-BASIC sau GW-BASIC de exemplu).
La fel limbajele LOGO sau PASCAL.

Aceasta este forma finală de reprezentare a unui algoritm, în continuare apărînd doar probleme de utilizare.

EXERCIŢII

1. Să se rezolve prima problemă cînd avem două vase de 3 şi 8 litri pentru a obţine : a) 4 litri; b) 5 litri; c) 7 litri;

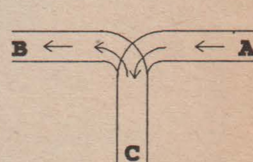
2. Într-un vas mare sînt 12 litri de suc. Cum pot fi împărţiţi ei în două părţi egale dacă nu avem la dispoziţie decît două vase de 5 şi 8 litri ? (Deci, în final trebuie să rămînă 6 litri în vasul mare şi 6 litri în vasul de 8 litri) .

3. Cîţi algoritmi se pot da în problema cu traversările dacă sînt permise a) cel mult 11 traversări; b) exact 11 traversări;

4. Să presupunem că se fac `2n+1` traversări. Cîte din acestea le face omul însoţit de capră ?

5. Să presupunem că într-un triaj există o linie de cale ferată pentru manevre, ca în figură. Pe linia **A** se află o garnitură cu vagoanele 1, 2, 3, 4 aranjate în această ordine. Pentru ca ele să fie scoase pe linia **B** în ordinea 2, 4, 3, 1, acarii pot efectua succesiv următoarele mişcări:

1: **A**---→ **C** (se deplasează vagonul 1 din **A** în **C**)
2: **A**---→ **C** (se deplasează vagonul 2 din **A** în **C**)
2: **C**---→ **B** (se deplasează vagonul 2 din **C** în **B**)
3: **A**---→ **C** (se deplasează vagonul 3 din **A** în **C**)
4: **A**---→ **C** (se deplasează vagonul 4 din **A** în **C**)
4: **C**---→ **B** (se deplasează vagonul 4 din **C** în **B**)
3: **C**---→ **B** (se deplasează vagonul 3 din **C** în **B**)
1: **C**---→ **B** (se deplasează vagonul 1 din **C** în **B**)



Dacă la intrare se află o garnitură cu vagoanele 1, 2, 3, 4, 5, 6 este posibil ca la ieşire să fie formată o garnitură cu vagoanele în ordinea 3, 2, 5, 6, 4, 1 ? Cum ?

I.2 Proprietăţi generale

Am definit intuitiv un algoritm ca fiind o metodă folosită pentru rezolvarea unei probleme. Algoritmi există poate de cînd există civilizaţia umană. Orice activitate se supune unui algoritm. Pentru a semăna şi recolta, pentru a creşte vite, pentru a face o casă, trebuiesc urmate anumite reguli .

Denumirea a venit însă destul de tîrziu; în anul 825, un matematician persan - *Abu Ja'far ibn Musa al Khowarizmi*, a scris o carte de exerciţii. Titlul cărţii, *Kitab al-jabr wa'l-muqabala*, a introdus în matematică denumirea de **algebră**, iar numele autorului - cea de **algoritm**. Literar, "al Khowarizmi" semnifică " din oraşul Khowarazm ". Astăzi, acest oraş se numeşte Khiva şi se află în Uzbekistan.

Cum termenul de algoritm este folosit în special în matematici, aproape peste tot prin algoritm se subînţelege de obicei un algoritm matematic.

Am văzut însă că algoritmul este o metodă de rezolvare a unei probleme - nu neapărat matematică - deşi exemplificările se fac în special pentru probleme matematice; aşa că termenul este mult mai cuprinzător.

Semnificaţia lui a devenit deosebit de importantă în informatică, unde algoritmul reprezintă ceea ce trebuie să facă un calculator pentru rezolvarea unei probleme. Aceasta îl diferenţiază de forme aparent echivalente ca proces sau tehnică.

Pentru o problemă dată, există un algoritm care să o rezolve ?

Aici sînt trei răspunsuri posibile:

a. Da ! şi se construieşte o soluţie algoritmică (algoritm).

b. Nu ! şi se demonstrează (destul de dificil) imposibilitatea de a rezolva problema printr-un algoritm.

c. Nu ştim dacă există ! S-ar putea ca problema dată să aibă o soluţie algoritmică, şi ea se caută.

Cele mai cunoscute astfel de probleme se numesc **conjecturi**.

Dar ce proprietăţi trebuie să aibă un algoritm pentru a fi considerat ca atare?

Să considerăm următoarele probleme :

1. Care sînt toate zecimalele numărului 10/3 ?

2. Care sînt toate zecimalele numărului π ?

3. Care sînt primele 5 zecimale ale numărului π ?

Pentru prima problemă, soluţia este simplă. Împărţim cele două numere şi găsim 3,(3). Deşi sînt o infinitate de zecimale, toate sînt egale, deci răspunsul este clar şi obţinut în urma aplicării unui algoritm: algoritmul de împărţire, după care se foloseşte noţiunea de fracţie periodică simplă.

La a doua problemă avem tot un număr infinit de zecimale; dar ele nu se supun nici unei reguli de periodicitate.

Deci, oricît de multe ar fi zecimalele pe care le calculăm, totdeauna vor rămîne numere care trebuiesc aflate !

Un algoritm care să determine aceste valori ar trebui să nu se oprească niciodată ! Şi atunci, cînd poate el da un răspuns la problemă ? Niciodată ! În această situaţie, ne situăm în cazul (b) al problemei, pentru care nu există algoritmi.

Pentru problema a treia în schimb, să presupunem că fiecare calcul de zecimală pentru π ia o unitate de timp; deci după 5 unităţi de timp putem obţine rezultatul final: 3.14159 .

Algoritmul va fi cel de calcul al acestor zecimale.

Rezumînd,

UN ALGORITM NECESITĂ UN TIMP FINIT DE CALCUL.

După un anumit număr de treceri prin paşii lui, algoritmul se încheie. În cazul cînd se intră în diverse drumuri infinite care nu conduc la pasul final, nu mai avem de-a face cu un algoritm ci cu o procedură .

De asemenea ,

UN ALGORITM ESTE GENERAL

În general nu este interesantă o metodă care să dea restul împărţirii lui 413 la 27. Dacă însă ea va da restul împărţirii `oricăror două numere întregi nenule`, atunci avem un algoritm.

TOATE OPERAŢIILE SÎNT DEFINITE ŞI EFECTIVE

O operaţie este definită dacă în momentul cînd se începe efectuarea ei, **toate** variabilele cu care lucrează au valori care să poată conduce calculele la un rezultat final.

Ce diferenţă există între afirmaţiile următoare ?

❶ Formula de rezolvare a ecuaţiei **a*x+b=0** este **x=-b/a**.

❷ Ecuaţia **2*x-7=0** are rădăcina reală **x=7/2**.

În prima situaţie avem o soluţie generală, adevărată pentru orice **a** şi **b** reale **a≠0**. Ea este **utilizată** de algoritm pentru a obţine soluţia ecuaţiei de gradul I; nu este obţinută ca rezultat final.

Algoritmul de rezolvare a ecuaţiei de gradul I poate funcţiona numai cînd se ştiu efectiv valorile lui **a** şi **b** (deci cînd se dă o ecuaţie particulară). Altfel, operaţia **-b/a** care dă soluţia acelei ecuaţii, nu se poate efectua.

Un rezultat algebric înseamnă el însuşi un algoritm; aplicarea lui înseamnă înlocuirea calculelor algebrice cu calcule aritmetice.

De exemplu, pentru aria unui triunghi se pot da mai multe formule.

Fiecare formulă înseamnă un algoritm de calcul al ariei.

Aplicarea algoritmului se poate face din momentul cînd toate variabilele formulei respective se înlocuiesc cu valori.

EXERCIŢII

1. Se dă o mulţime infinită **A**. Se poate da un algoritm care să decidă dacă un element **a** este în mulţimea **A** ?

2. Pentru care din următoarele probleme se pot da algoritmi ?

- Să se determine primele **n** numere prime;
- Să se determine al 64-lea număr prim;
- Să se genereze toate numerele prime pînă la 100 000;
- Să se genereze toate numerele prime;

3. Se poate da un algoritm care să determine toate persoanele dintr-un oraş care au aceeaşi zi de naştere ?

4. Există în Bucureşti două persoane care au acelaşi număr de fire de păr în cap ? De ce ?

5. Se dă mulţimea

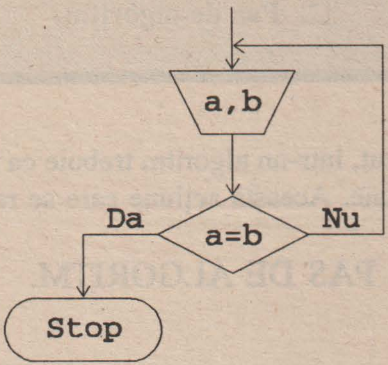
$$\mathbf{M} = \{(a, b, c, n) \setminus a, b, c, n \in \mathbb{N},\ n > 2,\ a^n + b^n = c^n\}$$

şi $x=0$ dacă $\mathbf{M} = \emptyset$, $x=1$ dacă $\mathbf{M} \neq \emptyset$.

Se poate da un algoritm care să-l determine pe x ?

6. Descrieţi sub formă de algoritm drumul de acasă pînă la şcoală.

7. Următoarea schemă logică desemnează sau nu un algoritm ?



8. Consideraţi că se poate trata drept algoritm:

- o prognoză meteorologică ?
- o reţetă culinară ?
- un tratament medical ?
- un meci de fotbal ?

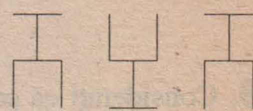
9. Se dă secvenţa de litere   I F M A M I I A
Să se găsească o metodă pentru a determina litera care urmează. Este această metodă un algoritm ?

1.3 Pas de algoritm

După cum am văzut, într-un algoritm trebuie ca la fiecare moment să fie executată o anumită acţiune. Această acţiune care se realizează se numeşte

PAS DE ALGORITM.

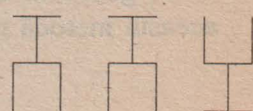
Să studiem următoarea problemă:



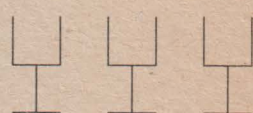
Se dau 3 pahare aşezate ca în figură.
*Numim mişcare acţiunea prin care, cu cele două mîini luăm două pahare şi le întoarcem simultan. Să se descrie cum pot fi aduse toate cele 3 pahare cu gura în sus, folosind **n** mişcări(**n** este un număr cunoscut).*

După un studiu atent, observăm că putem rezolva problema dată în oricîte etape vrem; să arătăm o soluţie pentru cazul **n**=2.

① se întorc paharele 2 şi 3;
Ajungem astfel la configuraţia



② se întorc paharele 1 şi 2
şi se obţine soluţia finală.



Algoritmul dat mai sus se efectuează în 2 paşi de algoritm. Aici, fiecare pas este compus dintr-o singură acţiune (mişcare).

Dacă facem însă o analiză mai detaliată a mişcării, ea este compusă din mai multe acţiuni care se pot executa una după alta sau simultan.

Astfel, pentru a obţine o mişcare (un pas de algoritm) este necesar ca:

$\boxed{a}$ Să apucăm un pahar cu mîna dreaptă

$\boxed{b}$ Să apucăm alt pahar cu mîna stîngă

$\boxed{c}$ Să întoarcem în acelaşi timp paharele

deci pasul 1 al algoritmului poate fi "rafinat" astfel:

$$\begin{array}{ccc}(a-2) & & (b-3)\\ & \searrow \quad \swarrow & \\ & (c) & \end{array}$$

unde am notat: (a-2) - operaţia a. efectuată cu paharul 2;
(b-3) - operaţia b. efectuată cu paharul 3.

Aceste două operaţii sînt independente una de alta şi deci pot fi executate simultan sau în paralel. În mod sigur însă, operaţia c. - de întoarcere a paharelor - se face **după** ce au fost efectuate celelalte două operaţii.

Acum putem scrie şi varianta mai detaliată a algoritmului, care rezolvă problema în două etape:

1. (a-2) , (b-3) , (c)
2. (a-1) , (b-2) , (c)

Rezumînd, vom înţelege prin pas de algoritm,

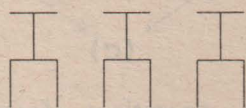
O SECVENŢĂ FINITĂ DE OPERAŢII (ACŢIUNI) CARE SE POT EFECTUA ÎNTR-O UNITATE STABILITĂ DE TIMP. UN PAS TREBUIE SĂ CONŢINĂ CEL PUŢIN O ACŢIUNE.

Exemplificînd pe limbajul de programare BASIC, o linie este un pas de algoritm; acest pas este definit prin numărul asociat liniei respective.

EXERCIŢII

1. Să se arate că pentru orice `n>0` există un algoritm de rezolvare a problemei anterioare în `n` paşi.

2. Este posibilă problema pentru cazul cînd paharele sînt aşezate iniţial



De ce ?

3. Se dă expresia aritmetică `a+b+x*y` . Să se arate ordinea de efectuare a operaţiilor în

a. 3 paşi de algoritm;

b. 2 paşi de algoritm;

Se presupune că într-o unitate de timp, o variabilă poate fi prelucrată cel mult odată.

4. Aceeaşi problemă pentru expresiile aritmetice :

`(a+b)/(x+y*z)  ,  x=a-2/(b+c*x*(a+y+z)-5);`

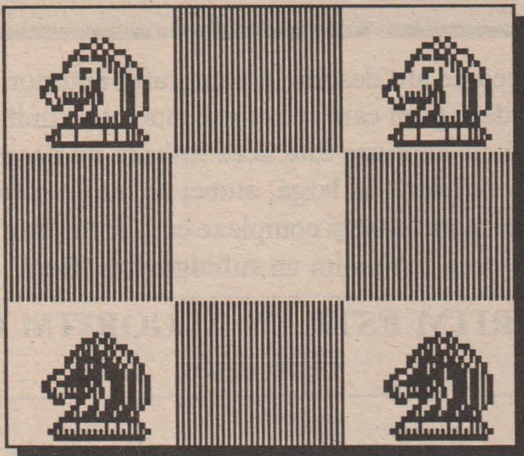
Care este numărul minim de paşi în care se pot rezolva ?

5. Într-un atelier se află 3 muncitori. Într-o unitate de timp:

- primul muncitor (`a`) face un şurub;
- al doilea muncitor (`b`) face o piuliţă;
- al treilea muncitor (`c`) le asamblează şi le pune într-un plic sigilat împreună cu un preţ de vînzare ;

Dimineaţa nu există nici o piesă făcută, iar la sfîrşitul programului nu rămîn piese disparate. Să se descrie algoritmul de lucru al celor 3 muncitori.

6. Pe tabla de dimensiune 3X3, de mai jos, se află doi cai albi şi doi cai negri.



Să se aducă cei doi cai albi pe poziţiile cailor negri şi reciproc, folosind numai mutări permise calului în jocul de şah.

7. Se dau următoarele numere de cinci cifre

25134

14253

Să se afle un număr avînd cifrele 1,2,3,4,5 (ca mai sus) astfel ca să aibă două poziţii comune cu primul număr şi trei poziţii comune cu al doilea număr.

1.4 Subalgoritm

Rezumînd ceea ce am descris în paragraful anterior, un pas de algoritm cuprinde o secvenţă de acţiuni care se pot executa într-o unitate de timp. Nicăieri însă nu se precizează **cît de mare este acea unitate de timp**.

Dacă ea este suficient de largă, atunci acţiunile care compun un pas de algoritm sînt destul de numeroase şi complexe ca să formeze ele însele un algoritm complet. Acesta este ceea ce numim un **subalgoritm**. Deci

UN SUBALGORITM ESTE UN ALGORITM CARE OCUPĂ UN PAS.

În exemplul anterior - cu paharele - fiecare pas de algoritm separat este un subalgoritm format la rîndul său din 3 paşi (a, b, c).

Un subalgoritm este foarte util dacă doi sau mai mulţi paşi de algoritm conţin aceeaşi secvenţă de operaţii, **diferenţa fiind doar de valorile cu care lucrează acestea.**

Atunci este de preferat să scriem subalgoritmul o singură dată - separat- precizînd numai valorile necesare executării lui.

De exemplu, pentru algoritmul în doi paşi dat anterior, putem construi subalgoritmul **întoarce(x, y)** astfel:

întoarce(x, y)

$\boxed{a}$ se ia paharul **x** cu mîna dreaptă

$\boxed{b}$ se ia paharul **y** cu mîna stîngă

$\boxed{c}$ se întorc simultan paharele **x** şi **y**.

Acum, algoritmul de rezolvare a problemei cu paharele se poate scrie:

1. **întoarce**(2,3)

2. **întoarce**(1,3)

La pasul 1 se efectuează paşii a - b - c în care **x** este înlocuit cu 2 şi **y** cu 3, iar la pasul 2, aceiaşi paşi sînt parcurşi pentru 1 (pe poziţia lui **x**) respectiv 3 (pe poziţia lui **y**).

Dacă faptele reale nu corespund teoriei,
trebuie să scăpăm de ele.
(Legea lui Meyer)

II. LIMBAJUL ALGORITMIC PSEUDOCOD
II.1 Generalităţi

După cum am văzut, procesul de rezolvare a unei probleme cuprinde mai multe etape. Avem astfel:

❶ Aflarea unui algoritm care să rezolve problema
❷ Transcrierea algoritmului într-un limbaj de programare
❸ Verificarea corectitudinii rezultatelor programului (validarea algoritmului)

Deosebirea esenţială între primii doi paşi este modalitatea de reprezentare a algoritmului. Să presupunem că se întîlnesc două persoane care deţin rezolvarea aceleiaşi probleme, scrisă în limbajele PASCAL şi respectiv BASIC. Prima persoană ştie numai PASCAL fără a avea nici cea mai mică idee despre programarea în BASIC iar a doua persoană n-a auzit în viaţa ei de PASCAL, cunoştinţele ei de BASIC fiind considerate suficiente. Cum îşi pot compara cele două persoane programele, să vadă care soluţie este mai bună?

O idee ar fi ca una din ele să cedeze şi să înveţe limbajul cunoscut de partener. Este o metodă coercitivă ! Învingătorul îşi impune cu forţa propriul său limbaj.

O altă rezolvare posibilă este ca cele două persoane interesate să-şi stabilească o limbă comună în care să-şi transmită algoritmii. Odată stabilit un astfel de limbaj universal - independent de calculatorul folosit - comunicarea se poate face uşor între orice comunităţi care folosesc orice limbaj, nu numai PASCAL şi BASIC. Se pot stabili contacte şi cu utilizatori de LOGO, LISP, DBASE şi aşa mai departe - fapt remarcabil dacă ţinem cont că pe glob se folosesc cîteva sute de limbaje de programare.

Aceasta este o metodă diplomatică de înţelegere izvorîtă din observaţia că în cele cinci decenii care au trecut de la naşterea limbajelor de programare, nici unul nu a putut cuceri o supremaţie totală nici în timp şi nici în profunzimea masei de utilizatori.

Deci s-a căzut de acord cu definirea unei reprezentări unitare a algoritmilor, reprezentare din care ulterior fiecare să-şi poată scrie un program în propriul său limbaj. O astfel de reprezentare se întîlneşte şi în momentul cînd se învaţă un limbaj (BASIC de exemplu): este vorba de **schemele logice**.

Ele sînt folosite şi acum destul de mult dar cu reţinere din cauza a două deficienţe majore:

◆ Într-o schemă logică se dă egală importanţă detaliului ca şi componentelor principale ale algoritmului. Cum în general detaliile abundă, ele ajung să sufoce algoritmul şi astfel el este greu de urmărit.

◆ Programele mari sînt în general structurate pe componente (numite **module**) avînd între ele diverse legături (uneori, de exemplu, un modul se poate apela pe el însuşi). Acest fapt este practic imposibil de prins într-o schemă logică.

•

De aceea treptat s-a impus o altă modalitate de reprezentare a algoritmilor - mai apropiată de programare. De fapt ea este derivată dintr-un limbaj vechi numit ALGOL (ALGOrithmic Language) care prin transformare treptată a ajuns să fie folosit sub denumirea de **pseudocod** numai în acest scop.

Deci:

Un limbaj **pseudocod** este un cadru de reprezentare a soluţiilor unor probleme formulate într-un anumit limbaj (matematic, natural, etc), permiţînd transcrierea lor ulterioară într-un limbaj de programare real (BASIC, LOGO, C, PASCAL, etc).

Un limbaj **pseudocod** traduce problema adusă pentru rezolvare într-o succesiune de acţiuni numite `operaţii` sau `instrucţiuni`.

În general fiecare pas de algoritm conţine o `operaţie`.

Cînd într-un pas de algoritm sînt cuprinse mai multe `operaţii`, acestea se despart prin $\boxed{;}$.

Orice text care nu formează un pas de algoritm este o **declaraţie** sau un **comentariu**.

EXERCIŢII

1. Pentru un număr n dat să se scrie în BASIC şi în LOGO programe care să traseze pătrate de latură n. Ce diferenţe există în scrierea acestor două programe ? Dar asemănări ?

2. Ce limbaj de programare preferaţi pentru:
- Calculul sumei a două numere
- Desenarea unei hărţi

3. Ce elemente consideraţi că trebuie să conţină un limbaj de programare care să poată citi portative cu note şi să compună muzică ?

II.2 Comentarii

Un comentariu poate fi plasat oriunde în algoritm şi foloseşte la explicarea operaţiilor.

Faptul că avem de-a face cu un comentariu se semnalează prin plasarea sa între caracterele `/*` la început şi respectiv `*/` la sfîrşit. Astfel,

/* Acesta este un comentariu */

reprezintă un text a cărui apariţie nu este executată de algoritm, dar explică faptul că avem de-a face cu un comentariu.

II.3 Declarări

Toţi paşii unui algoritm (deci toate operaţiile) prelucrează date. Aceste date sînt în general `constante` sau `variabile`.

`Constantele` sînt în majoritatea lor ceea ce înţelegem în matematică prin numere - 2, 0, 1, 3.1415..., dar şi litere ale alfabetului- a, b,.., sau două cuvinte speciale a căror importanţă o vom semnala mai tîrziu : **adevărat, fals.**

Precizăm : acestea nu sînt **toate** tipurile posibile de constante; odată cu creşterea complexităţii algoritmilor vor fi introduse şi altele.

Deci, vom prelucra constante

① **numerice**

② **alfabetice**

Adesea, aceste două tipuri de constante se numesc împreună constante **alfanumerice.**

③ **logice**: constantele **adevărat** şi **fals.**

Pentru constante nu este necesară o specificare specială. Ele pot apare oriunde în algoritm aşa cum sînt (pentru constantele **numerice** sau **logice**) sau încadrate de apostrofuri la cele **alfabetice** (**'a'**, **'b'**,..).

`Variabilele`; aşa cum se ştie de la limbajele BASIC şi LOGO, ele sînt zone de memorie în care se depun constante; aceeaşi semnificaţie va fi dată şi în cazul **pseudocodului.**

FIECARE VARIABILĂ CONŢINE UN SINGUR TIP DE CONSTANTE

Acest tip **trebuie declarat.**

Avem astfel de-a face cu variabile de tip **întreg, real** (dacă în ele se depun constante întregi respectiv reale), **caracter** (dacă se depun constante aflate pe tastatura maşinii de scris) sau **logice** (variabile în care se poate păstra una din cele două constante logice definite anterior).

Aceste tipuri se anunţă obligatoriu printr-o instrucţiune numită `declarare` care nu aparţine nici unui pas de algoritm.

O VARIABILĂ NU POATE FI FOLOSITĂ ÎN ALGORITM DACĂ NU A FOST DECLARATĂ ANTERIOR.

De aceea declarările se fac de obicei **înainte de primul pas al algoritmului.**

Exemplu:

```
a,b,i  întregi
x,y reale
p logic
```

anunţă că în algoritmul care urmează se va lucra cu variabilele `a,b,i` în care se reţin valori întregi, `x,y` - variabile reale şi `p` - o variabilă în care nu se pot plasa decît valorile **adevărat** sau **fals.**

Spunem că `a,b,i` sînt de tip **`întreg`**, `x,y` de tip **`real`** iar `p` - de tip **logic**.

O variabilă nu poate fi declarată de două ori în acelaşi algoritm, nici de acelaşi tip şi nici de tipuri diferite.

Un tip special de declarare este acela al numelui algoritmului.

Orice algoritm se identifică (aşa cum se identifică persoanele) printr-un nume scris la începutul algoritmului. Numele poate fi urmat eventual şi de o listă de variabile numite **parametri**.

Exemplu:

```
polinom , euclid(a,b) , întoarce(x,y)
```

Lista de parametri se utilizează în general atunci cînd se declară un subalgoritm, aşa cum a fost subalgoritmul **întoarce** prezentat anterior.

III.4 Operaţiunea de oprire

În schemele logice se notează prin (Stop) şi marchează sfîrşitul algoritmului.

Exemplu:

```
algoritm stop
/* Algoritmul se opreşte la primul pas */
1. Stop
```

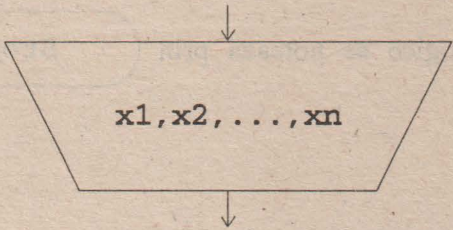
Dacă ne aflăm într-un subalgoritm şi vrem să ne întoarcem în algoritmul care apelează, în loc de `Stop` se foloseşte `Revenire`. Dacă algoritmul se opreşte în subalgoritm, aceasta se realizează evident tot cu `Stop`.

```
algoritm stop1
1. sub      /* se face apel la subalgoritmul sub */
2. Stop     /* algoritmul stop1 se opreşte */
sub
1. Revenire /* sub se închide şi se revine în stop1 */
```

```
algoritm stop2
1. sub   /* se face apel la subalgoritmul sub */
2. Stop  /* operaţia de oprire */
sub
1. Stop  /* se încheie simultan sub şi stop2 */
```

III.5 Operaţii de intrare/ieşire

Operaţia de intrare, reprezentată în schema logică prin



este definită

`citeşte x1, x2 , ... , xn`

unde x1 , x2 ,..., xn sînt nume de variabile.

Prin această operaţie, în cele n variabile **xi** se introduc valori pe care algoritmul le va folosi ulterior.

Se consideră că după acest pas, variabilele x1 , x2 ,..., xn **sînt definite.**

Observaţii:

♦ Valorile citite în variabile trebuie să fie în concordanţă cu tipurile prin care au fost introduse aceste variabile.

♦ O variabilă este considerată **definită** după ce a fost declarată şi după ce printr-o modalitate oarecare, în ea s-a depus o valoare.

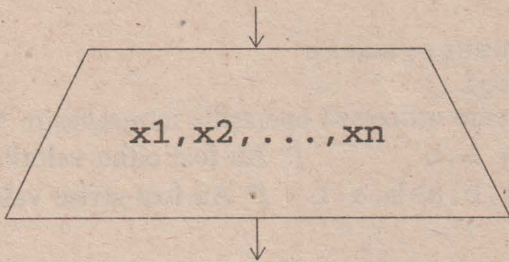
Pentru a elimina eventuale confuzii, o definiţie generală se scrie

citeşte < lista de variabile >

în care **citeşte** este un cuvînt caracteristic operaţiei de intrare (numit cuvînt cheie), iar < lista de variabile > este o secvenţă de variabile separate prin virgule (de forma x1, x2, ..., xn).

♦ În lista de variabile există cel puţin o variabilă.
♦ Toate variabilele din listă sînt distincte una de alta.

Similar, pentru operaţia de ieşire avem reprezentarea în schema logică



În pseudocod, definiţia este

scrie `< lista de variabile >`

unde `<lista de variabile >` are aceeaşi semnificaţie de la operaţia de citire. Aici însă variabilele din listă reprezintă **rezultate** şi ele pot fi astfel puse la îndemîna celui care utilizează algoritmul.

De remarcat că pentru uşurinţa înţelegerii rezultatelor, în această listă pot fi inserate şi texte de tip **comentariu** care, spre deosebire de textele **comentariu**, apar la ieşire. Cum ele aparţin însă unui pas de algoritm vor fi încadrate de ’.

Exemplu

`scrie` ’Rezultatul este ’, `x`

Dacă în variabila **x** se află valoarea 2, la ieşire va apare textul

Rezultatul este 2

Instrucţiunile de intrare/ieşire reprezintă de fapt cele două capete ale schemei de la pagina 9.

Exemplu

Algoritmul următor introduce în variabilele **a** şi **b** două numere şi scrie (afişează, scoate) aceste numere precum şi suma şi diferenţa lor;

```
algoritm citeşte/scrie
a,b întregi
/* Algoritm care utilizează operaţiile intrare/ieşire */
1. citeşte a,b          /* Au fost citite valorile lui a şi b */
2. scrie a,b,a+b,a-b    /* Au fost scrise valorile rezultate */
3. Stop
```

EXERCIŢII

1. Ce deosebire este între constantele 1 şi '1' ?

2. Cum este bine : 1. `citeşte x`
2. `scrie` 'Valoarea lui **x** este ',`x` , sau
1. `citeşte x`
2. `scrie` 'Valoarea lui **x** este ,`x` '

3. Este corect algoritmul

```
a întreg
 1. scrie a
 2. citeşte a
 3. Stop
```

Dar

```
a întreg
 1. citeşte a; scrie a
 2. citeşte a; scrie a
 3. Stop
```

4. Se dă algoritmul

```
algoritm trecere
    a  întreg
     1. citeşte a
     2. copie (a)
     3. Stop
    copie (a)
    a întreg
     1. scrie a
     2.  XXX
```

Cum lucrează acest algoritm dacă în loc de XXX apare operaţia
i) `Stop` ii) `Revenire`

III.6 Expresii aritmetice

În scrierea algoritmilor, deosebit de mult folosite sînt expresiile aritmetice. Utilizarea lor, deşi cunoscută, necesită totuşi cîteva precauţii impuse de restricţiile scrierii pe calculator. Astfel:

a Componentele unei expresii sînt:

- **operanzi**: variabile sau constante numerice.

  Orice expresie aritmetică conţine cel puţin un operand.
- **operatori binari** : + , - , * , / ,^ .

  Ei utilizează doi operanzi pe care îi prelucrează dînd un rezultat.

  Ca o remarcă, exponenţierea ^ , deşi este operator binar, nu există în toate limbajele de programare.
- **operator unar** : - .

  Este semnul - care poate apare în faţa operanzilor, ca de exemplu -5 , -(x+y). În expresia **-a-b**, primul - este unar iar al doilea, binar.
- **paranteze** : (,).

  În interiorul unei perechi de paranteze se află totdeauna o expresie aritmetică. Nu se folosesc alte tipuri de paranteze ca [,] , { sau }.
- diverse **funcţii** considerate **standard**; alegerea lor este în general dependentă de limbaj. Vom considera totuşi cîteva funcţii standard folosite universal.

`abs` definită

`abs(x)=x` dacă `x≥0`
`abs(x)=-x` dacă `x<0`

`int`;

pentru **x** real, `int(x)` păstrează cel mai mare număr întreg cuprins în **x** (partea întreagă a lui **x**).

`rădăcina`;

`rădăcina(x)` este acel număr pozitiv **y** cu proprietatea $y^2=x$.

`sin,cos,tan,asin,acos,atan` - funcţii trigonometrice.

`ln` - funcţie logaritmică, etc.

ORICE OPERATOR SE SCRIE EXPLICIT ÎN EXPRESIE.

O exprimare de forma **a(b+c)** este greşită. Corect se scrie **a*(b+c)**.

b Liniarizare.

În scrierea unei expresii aritmetice, reprezentări ca de exemplu a^3, $\frac{x}{y+z}$ nu sînt permise.

ORICE EXPRESIE ARITMETICĂ SE SCRIE LINIAR, PE UN RÎND.

Cele două exemple de sus se exprimă corect prin **a^3** respectiv **x/(y+z)**.

Exprimarea liniară a expresiilor aritmetice nu este totdeauna uşoară; pentru a o stăpîni bine, trebuie să ţinem cont de aşa numitele priorităţi de calcul.

c Priorităţi de calcul .

Prelucrarea expresiilor aritmetice se face totdeauna de la stînga la dreapta, conform următoarelor convenţii:

- se execută întîi funcţiile standard;
- urmează prelucrarea expresiilor aritmetice din paranteze;
- se execută operatorul unar -;
- se execută exponenţierile;
- se execută înmulţirile şi împărţirile;
- se execută adunările şi scăderile.

DACĂ DOI OPERATORI DE ACEEAŞI PRIORITATE SÎNT CONSECUTIVI, EI SE EXECUTĂ ÎN ORDINEA APARIŢIEI LOR - DE LA STÎNGA LA DREAPTA - CU EXCEPŢIA EXPONENŢIERILOR.

Exemple

i. a+b-c

Ordinea de execuţie: 1. adunarea lui **a** cu **b**;

2. din rezultatul obţinut în 1. se scade **a**.

ii. a+(b+c)

1. se adună **b** cu **c**;
2. **a** se adună cu rezultatul de la 1.

iii. Să se liniarizeze expresia aritmetică

$$\frac{a+b+c}{x^2+y^2}-5*k$$

Liniarizarea se face ţinînd cont şi de ordinea de execuţie a operaţiilor:

(a+b+c)/(x^2+y^2)-5*k

iv. Care este ordinea de efectuare a operaţiilor în expresia aritmetică

a*(b-x/2*rădăcina(b)) ?

1. **rădăcina(b)**;
2. **x/2**;
3. rezultatele de la 1 şi 2 se înmulţesc;
4. se scade din **b** rezultatul de la 3;
5. **a** se înmulţeşte cu rezultatul de la 4.

Se observă că acest exemplu este scrierea liniarizată a expresiei

$$a*(b-\frac{x\sqrt{b}}{2})$$

[d] Tipul operatorilor.

Am văzut că variabilele şi constantele numerice dintr-o expresie aritmetică se consideră de două tipuri: **întreg** şi **real**.

Datorită faptului că aceste tipuri de operanzi se reţin în calculator în mod diferit, vor diferi şi operaţiile care se fac cu ei.

Astfel, orice operaţie aritmetică **a @ b** se supune următoarei reguli:

DACĂ a ŞI b SÎNT ÎNTREGI, ATUNCI REZULTATUL OPERAŢIEI a @ b ESTE ÎNTREG. DACĂ CEL PUŢIN UNUL DIN OPERANZI ESTE REAL, ATUNCI REZULTATUL ESTE REAL.

Afirmaţiile par simple în cazul operatorilor + , - şi * , diferenţele nefiind sesizabile la nivel teoretic.

În cazul împărţirii, se desprinde ca un caz particular regula:

DACĂ a ŞI b SÎNT ÎNTREGI, ATUNCI a/b ESTE ÎNTREG.

De exemplu, 5/2 este întreg; valoarea lui este 2.
De fapt, în această situaţie are loc egalitatea **a/b = int(a/b).**
În schimb

5./2 = 5/2. = 5./2. = 2.5

ORICE VALOARE ÎNTREAGĂ DEPUSĂ ÎNTR-O VARIABILĂ REALĂ DEVINE REALĂ. ORICE VALOARE REALĂ DEPUSĂ ÎNTR-O VARIABILĂ ÎNTREAGĂ, DEVINE ÎNTREAGĂ.

Deoarece în calculator mulţimile de numere nu sînt nemărginite, ci situate în intervale finite (ca de exemplu [-32 768 , 32 767] la numerele întregi) în faza de aplicare a algoritmului pot exista erori generate de depăşirea acestor intervale.

Exemplu

Să presupunem că avem operaţia **a*b**, cu ambii operanzi întregi. Algoritmul este transpus sub formă de program şi executat. În această fază, dacă avem o prelucrare cu valorile 10.000 în **a** şi 20.000 în **b**, rezultatul nu este 200.000.000 ca cel teoretic, ci - depăşind domeniul - va genera o eroare de calcul.

Cele mai frecvente astfel de erori pentru variabilele reale sînt cele generate de împărţirea **a/b**, cînd **b** ia valori foarte mici; atunci **a/b = a*(1./b)** şi cum **1./b** devine foarte mare, înmulţirea va genera depăşire.

EXERCIŢII

1. Să se liniarizeze expresiile aritmetice:

$$\frac{a}{2\left(\frac{x+y}{b}+\frac{x+z}{c}\right)}-\frac{b}{a+1.3} \qquad \frac{\frac{x}{abc}+\frac{y}{a+b+c}+\frac{2}{ab+bc+ac}}{xyz}$$

$$\sqrt{\frac{ab}{\sqrt{c}}-\frac{bc}{\sqrt{a}}+\frac{ac}{\sqrt{b}}} \qquad 3^{\sqrt{a}}*a^{\sqrt{3}} \qquad \frac{\sqrt{abs(a)}}{a}-\frac{\sqrt{abs(b)}}{b}$$

2. În ce ordine se execută operaţiile în următoarele expresii aritmetice ?

rădăcina(b*b-4*a*c)
(x+y/z/2.5)^(a-abs(a)*cina(x*y-43))
sin(a*cos(x+y)+b*cos(x-y))

3. Toate operaţiile de mai jos sînt cu numere întregi. Să se afle valoarea expresiilor:

1/2+1/2+1/2;	**70/3/2;**	**104/7/5;**	**8/3;**	**8*1/3;**	**1/3*8;**
(5*3-2*4)/2/2;	**3^2^2;**	**2^3^4;**	**3^2/3;**		

III.7 Operaţia de atribuire (asignare)

Această instrucţiune este fundamentală în scrierea algoritmilor. În cadrul schemelor logice se folosea

```
   ↓
| v<---e |
   ↓
```

iar în pseudocod, forma generală este pur şi simplu

v ←-- - e

unde v este nume de variabilă din lista tipurilor scrise la începutul algoritmului, iar e poate fi:

- o expresie aritmetică avînd ca rezultat o valoare de tip întreg sau real, care se transformă în tipul variabilei v şi se depune la adresa acesteia;
- un caracter, dacă v este de tip caracter;
- o valoare logică, dacă variabila v este de tip **logic**.

De exemplu

a ←--- 2+5 are ca efect atribuirea valorii **7** variabilei **a**;

x ←--- 'p' : variabila **x** a fost declarată de tip caracter şi prin această operaţie, ei i s-a atribuit valoarea **'p'**.

După o operaţie de atribuire, variabila din membrul stîng devine definită.

Observaţii:

♦ Operaţia de intrare lucrează similar operaţiei de atribuire. Astfel, în secvenţa

```
a real; b caracter ; c logic
citeşte a,b,c
```

algoritmul aşteaptă să dăm 3 valori conforme cu tipul celor 3 variabile.

Dacă am dat

```
5.2   x   adevărat
```

atunci operaţia de citire este echivalentă cu secvenţa de atribuiri

```
a ←--- 5.2 ; b ←--- 'x' ; c ←--- adevărat
```

De remarcat că introducerea valorilor în altă ordine este greşită; ea **trebuie să corespundă cu ordinea variabilelor din listă** (şi deci şi cu ordinea tipurilor lor).

♦ Un algoritm apelează un subalgoritm **sub(a,b)**; în momentul cînd se ajunge la acest pas, variabilele **a** şi **b** sînt definite.

Se trece imediat la parcurgerea subalgoritmului **sub**, al cărui nume este **sub(x,y)**. Primul lucru care se efectuează în acest moment este secvenţa de atribuiri

```
x ←--- a ; y ←--- b
```

Exemplu

```
atribuire
a,b reale
1. citeşte a   /* Se citeşte în a o valoare reală */
2. b ←--- a    /* Valoarea lui a se copiază şi în b */
3. scrie b     /* Se scrie valoarea aflată în b */
4. Stop
```

Dacă înlocuim pasul 2 cu

```
2'. b ←--- a+b
```

operaţia este greşită deoarece acum variabila **b** din membrul drept nu este definită.

Exemplu

```
identic
a caracter
  1. citeşte a
  2. a ←--- a
  3. scrie a
  4. Stop
```

Algoritmul este corect, dar nu efectuează de fapt nimic, atribuirea `a ←--- a` introducînd în **a** tot valoarea lui **a** .

Atenţie (!) Operaţia de atribuire comportă două momente distincte:

☞ În prima fază se evaluează expresia din dreapta. Toate variabilele aflate în compunerea ei trebuie să fie definite anterior (prin citire sau alte atribuiri). Se obţine astfel o valoare.

☞ În faza a doua, valoarea obţinută se depune la adresa variabilei din stînga operaţiei de atribuire (pentru noţiunile de **valoare** şi **adresă** se pot revedea orice cărţi de prezentare ale limbajelor BASIC, PASCAL sau LOGO).

Exemplu

```
contor
i întreg
1. citeşte i   /* Se dă o valoare lui i */
2. i ←--- i+1 /* Valoarea lui i se măreşte cu unitate */
3. scrie i     /* Se scrie noua valoare a lui i */
4. Stop
```

Deci, dacă prin citire s-a introdus în `i` valoarea 7, la scriere va apare valoarea 8. La acelaşi rezultat conduce şi algoritmul

```
increment
i întreg
  1. citeşte i ; scrie i+1
  2. Stop
```

Diferenţa constă în ceea ce rămîne la sfîrşit în variabila `i`; valoarea 8 la algoritmul `contor`, valoarea 7 (nemodificată) la algoritmul `increment`.

EXERCIŢII

1. Se citeşte o valoare în variabila `a`. Să se scrie valoarea lui `a` fără semn (dacă `a` ia valoarea -2, se va scrie 2).
Indicaţie : `a ←--- rădăcina(a*a)`

2. Ce efect au paşii de algoritm :

a)
```
1. citeşte a
2. a ←--- a-a
3. b ←--- a
4. scrie a,b
```
şi
```
1. citeşte a
2. b ←--- a
3. a ←--- a-a
4. scrie a,b
```

b)
```
1. citeşte a,b
2. a ←--- a+b
3. b ←--- a+b
4. scrie a,b
```
şi
```
1. citeşte a,b
2. b ←--- a+b
3. a ←--- a+b
4. scrie a,b
```

c)
```
1. citeşte a
2. b ←--- a
3. a ←--- a+b
```
şi
```
1. citeşte a
2. a ←--- a+b
3. b ←--- a
```

III.8 Expresii logice

În mod similar expresiilor aritmetice - al căror rezultat este o valoare numerică (de tip `întreg` sau `real`) - se pot construi şi expresii al căror rezultat este o valoare logică.

Sînt două tipuri de astfel de expresii:

a **expresii relaţionale**

de forma

< expresie aritmetică 1 > < relaţie > <expresie aritmetică 2 >

Aici, **< expresie aritmetică >** este cea definită anterior, iar **< relaţie >** este unul din operatorii

$<$ - mai mic
$\leq$ - mai mic sau egal
$>$ - mai mare
$\geq$ - mai mare sau egal
$=$ - egal
$\neq$ - diferit

Exemple

`2<i+1 ; k=3 ; x*x+y*y≥0 ; rădăcina(p)≠2 .`

Efectul unei astfel de expresii relaţionale este:

- Se evaluează cele două expresii aritmetice;
- Se cercetează dacă valorile lor se află în relaţia specificată între ele ;
- Rezultatul este valoarea **adevărat** sau **fals**, după decizia care se ia în urma comparaţiei.

Exemplu

Avem expresia relaţională `a*b+5>x.`

Dacă **a ←--- 0 , b ←--- 1 , x ←--- 1** , după evaluările celor două expresii aritmetice se obţine forma **5 > 1** care ia valoarea **adevărat.**

Dacă valorile care definesc variabilele sînt **a ←--- -1, b ←--- 5, x ←--- 0** atunci se ajunge la **0 > 0**, inegalitate **falsă.**

Expresia

a>b>0

nu este o expresie relaţională corectă, nici în această formă şi nici ca

(a>b)>0 sau a>(b>0)

Aceasta deoarece - conform construcţiei a. - o expresie relaţională conţine un singur operator de tip **<relaţie>.**

b expresii logice

sînt expresii construite cu:

1. constante şi variabile logice;
2. expresii care iau valori logice (expresii relaţionale);
3. operatorii logici nu,şi,sau.

Aceşti operatori lucrează numai cu valorile logice **adevărat**, **fals** în felul următor :

nu(**adevărat**)≡**fals** , nu(**fals**)≡**adevărat**

sau	**adevărat**	**fals**
adevărat	adevărat	adevărat
fals	adevărat	fals

şi	**adevărat**	**fals**
adevărat	adevărat	fals
fals	fals	fals

Deci rezultatul unei expresii logice este o valoare logică.

De remarcat că am folosit un semn nou ≡ în loc de = ; cînd scriem x≡y aceasta înseamnă că expresiile x şi y (aritmetice sau logice) iau simultan aceleaşi valori (numerice respectiv logice).

≡ poartă numele de **identitate** .

Exemple

`(a>b) şi nu (x≤2)` ; aici `a,b,x` sînt variabile numerice

`a sau b şi nu (c<0)` ; aici `a` şi `b` sînt variabile logice iar `c` - variabilă numerică.

`nu a nu (x>z)` este greşită, deoarece al doilea operator `nu` are doi operanzi, ceea ce intră în contradicţie cu definiţia sa de operator unar.

Ca şi în cazul expresiilor aritmetice, între operatorii relaţionali şi logici există priorităţi de calcul. Într-o expresie logică, ordinea de execuţie a operatorilor este:

- funcţiile standard
- expresiile între paranteze
- expresiile aritmetice (cu priorităţile cunoscute)
- expresiile relaţionale (de la stînga la dreapta)
- `nu`
- `şi`
- `sau`.

Exemplu

Fie expresia logică

`a+b*(x-y)>rădăcina(4) şi nu(2-a=3^x) or v`

Să presupunem că prin diverse operaţii (citire sau atribuire), variabilele au fost definite cu valorile

a ←--- 0 , b ←--- 1 , x ←--- 2 , y ←--- 1 , v ←--- fals

Ordinea de efectuare a calculelor este :

① **rădăcina(4) ≡ 2;**
② **x-y ≡ 2-1 ≡ 1;**
③ **3^x ≡ 3^2 ≡ 9;**
④ **2-a ≡ 2-0 ≡ 2;**
⑤ **a+b*(x-y) ≡ 0+1*1 ≡ 1;**

⑥ **2-a=3^x ≡ 2=9 ≡ fals;**
⑦ **a+b*(x-y) > rădăcina(4) ≡ 1>2 ≡ fals;**
⑧ **nu(2-a=3^x) ≡ nu(fals) ≡ adevărat;**
⑨ **a+b*(x-y) > rădăcina(4) şi nu(2-a=3^x) ≡ fals şi adevărat ≡ fals;**
⑩ **a+b*(x-y) > rădăcina(4) şi nu(2-a=3^x) or v=fals or fals ≡ fals.**

Deci valoarea întregii expresii este **fals.**

De remarcat diferenţa între:

= : operator relaţional care compară valorile din membrul stîng şi drept şi dă ca rezultat o valoare logică: **adevărat** dacă cele două valori sînt egale, **fals** altfel;

←--- : operaţia pseudocod de atribuire;

≡ : operaţia identitate ; expresiile din cei doi membri iau aceleaşi valori pentru orice definiri ale variabilelor care le compun.

EXERCIŢII

1. Pentru ce `x` şi `y` expresia `x*x+y*y=0` ia valoarea **adevărat** ?
Dar `x*x+x+1>0` ?

2. Să se scrie o expresie logică care să ia valoarea **adevărat** dacă şi numai dacă variabila întreagă `n` ia valori între 0 şi 120.

3. Să se scrie o expresie logică care să ia valoarea **adevărat** dacă şi numai dacă variabila `a` este un număr par mai mare decît -25.

4. Se dă expresia logică `nu(a='a')şi(a='z')or(a='x')`
Cînd este ea adevărată ?

5. Se dau două numere reale `x` şi `y`. Ce expresie relaţională ia valoarea **adevărat** dacă şi numai dacă:

- `x` şi `y` sînt de semne contrare

- x şi y sînt nenule
- cel puţin unul din ele este 0
- ambele sînt 0

6. Fie **a ←--- 0 , b ←--- 1 , x ←--- 2 , y ←--- adevărat**
Să se cerceteze dacă următoarele expresii logice sînt corecte şi în acest caz să se afle valorile lor:

```
a*b >1 or nu(x > 2 or b=1)
(a+b+x)>x şi y
(a+b+x)>(x şi y)
```

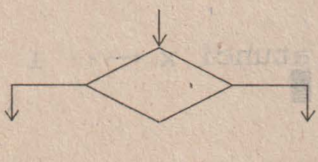
7. Să se arate că au loc identităţile:

```
a şi nu a ≡ fals;
a*a≥0 ≡ adevărat;
a≠b ≡ nu (a=b);
nu(a şi b) ≡ nu a sau nu b;
nu(a sau b) ≡ nu a şi nu b;
nu(a>b) ≡ a≤b;
a<b ≡ (a≤b) şi nu (a=b);
```

III.9 Operaţii de decizie

Folosind operaţiile pseudocod definite pînă acum, un algoritm se poate parcurge trecînd pe rînd de la un pas la altul, în ordinea apariţiei lor. O astfel de parcurgere se numeşte **secvenţială** sau **liniară.**

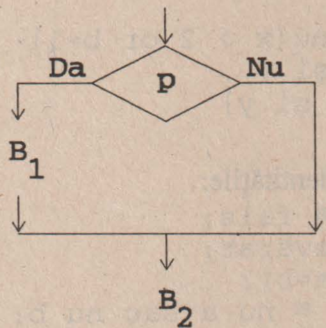
De multe ori însă anumite acţiuni trebuiesc executate numai în urma luării unor hotărîri. Pentru aceasta sînt necesare operaţii speciale numite **operaţii de decizie.** În schemele logice o astfel de operaţie este reprezentată



După variantele pe care le asigură cele două ramuri rezultante ale blocului de decizie, pseudocodul dezvoltă mai multe operaţii posibile.

Operaţii de decizie alternativă

a) pentru varianta



corespunde operaţia

dacă p atunci B_1
■
B_2

unde p este o expresie logică, iar B_1 şi B_2 - două secvenţe de algoritmi formate din unul sau mai mulţi paşi. Aceasta se numeşte operaţie de decizie alternativă **forma scurtă** şi prelucrarea ei se face în următoarele etape:

- se evaluează expresia logică p;
- dacă valoarea lui p este **adevărat**, se efectuează secvenţa de algoritmi B_1B_2.
- dacă valoarea lui p este **fals**, se efectuează secvenţa de algoritm B_2 (sărindu-se peste B_1).

Exemplu

```
x ←--- 0
dacă a>0 atunci x ←--- 1
         ■
scrie x
```

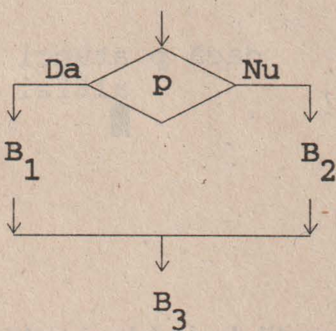
După ce se face testarea `a>0` (şi se află valoarea acestei expresii relaţionale) algoritmul are două ramuri posibile:

(a>0) ≡ adevărat. Atunci **x ←--- 1** şi operaţia de ieşire dă valoarea 1, sau

(a>0) ≡ fals. În acest caz **x** rămîne cu valoarea anterioară şi la ieşire va apare valoarea 0.

█ este un marcator care delimitează sfîrşitul secvenţei de algoritm B_1. Lipsa unui astfel de semn distinctiv ar face imposibilă separarea pasului de algoritm cînd se închide operaţia de decizie şi începe blocul B_2 .

ⓑ Operaţia generală de decizie alternativă, corespunzătoare schemei logice



este:

```
dacă p atunci B1
       altfel B2
       █
B3
```

Prelucrarea se face astfel:

- se evaluează expresia logică `p`;
- dacă valoarea lui `p` este **adevărat** , atunci se execută secvenţa de algoritmi B_1B_3 (fără B_2);
- dacă valoarea lui `p` este **fals,** se omite B_1 executîndu-se B_2B_3.

Exemplu

```
      dacă  a>0 atunci x ←--- 1
                altfel  x ←--- 2
                █
      scrie x
```

va scrie valoarea 1 dacă valoarea aflată în a este pozitivă, şi 2 dacă în a se află o valoare mai mică sau egală decît zero.

De remarcat că pentru blocul B_1 rolul de delimitator îl joacă cuvintele cheie **atunci** şi **altfel**, iar pentru B_2 - cuvîntul cheie **altfel** şi semnul █ .

Operaţia de decizie alternativă forma scurtă este un caz particular, cînd blocul B_2 nu conţine nici o operaţie pseudocod:

```
dacă p atunci B1
       altfel
       █
```

Operaţii de decizie repetitivă

Sînt introduse în secţiunea referitoare la algoritmii ciclici.

III.10 Operaţia de salt fără decizie

Nu este o operaţie esenţială, orice algoritm putînd - printr-o aranjare convenabilă a paşilor săi - să nu folosească astfel de instrucţiuni. Nu are echivalent în schemele logice.

Forma generală este

```
salt la k
```

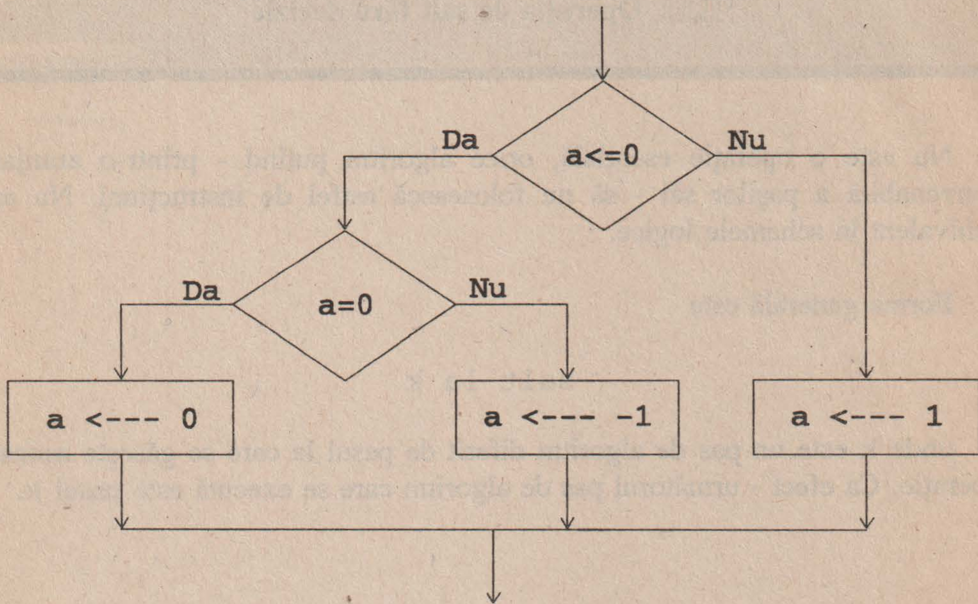
unde `k` este un pas de algoritm diferit de pasul la care se găseşte această operaţie. Ca efect - următorul pas de algoritm care se execută este pasul `k`.

EXERCIŢII

1. "Merg la şcoală în fiecare zi înafară de sîmbătă şi duminică ".
Cum se poate formaliza această propoziţie folosind operaţia de decizie alternativă ?

2. Aceeaşi problemă pentru propoziţiile:
" Cîinele Azor este singurul meu prieten " ;
" Fratele meu poate citi şi cărţi fără poze ".

3. Se dă schema logică:



Să se scrie operaţia de decizie alternativă corespunzătoare.

4. Cele două operaţii de decizie alternativă de mai jos diferă doar prin poziţia delimitatorilor. Ce diferenţe apar la evaluare ?

```
i.    dacă  x ≥ 0 atunci
                  dacă x=0 atunci y ←--- 1
                           altfel y ←--- 0
                           ■
            ■

ii.   dacă x ≥ 0 atunci
                  dacă x=0 atunci y ←--- 1
                           ■
           altfel y ←--- 0
           ■
```

Să se traseze schemele logice corespunzătoare.

5. Se dau următoarele secvenţe de operaţii pseudocod:

```
i.  dacă p atunci B1
          ■
    dacă nu p atunci B2
               ■
```

```
ii.  pp logic
     dacă p atunci pp←--nu p
                     B1
          ■
     dacă pp atunci B2
             ■
```

Se poate identifica vreuna din ele cu

```
dacă p atunci B1
altfel B2
■
```

De ce ?

În condiţii iniţiale riguros determinate,
un algoritm face ce-i place !
(Murphy)

III. CLASIFICAREA ALGORITMILOR
III.1 Algoritmi liniari

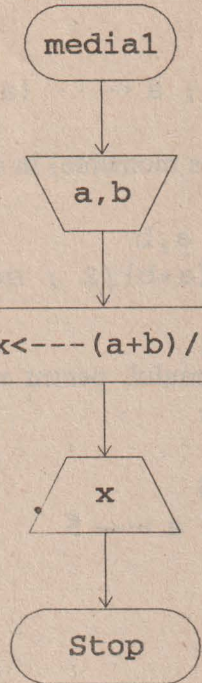
Aceştia constituie cea mai simplă clasă de algoritmi, neavînd nici un fel de operaţie de decizie. Execuţia lor se face secvenţial, într-o singură trecere; de aceea, de cele mai multe ori ei apar ca subalgoritmi în structuri mai complexe.

În reprezentarea sub formă de schemă logică, de la blocul (nume) pînă la (Stop) există un singur drum posibil.

1.1 Media aritmetică a două numere

```
media_1
x,a,b reale
1. citeşte a,b
2. x ←-- (a+b)/2
3. scrie x
4. Stop
```

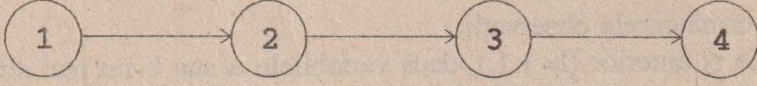
Dacă vrem să descriem algoritmul printr-o schemă logică, atunci



Dacă variabilele a şi/sau b nu sînt utilizate în altă parte a algoritmului, atunci putem folosi pe oricare din ele în locul lui x:

```
media_2
a,b reale
1. citeşte a,b
2. a ←-- (a+b)/2
3. scrie a
4. Stop
```

Ordinea de parcurgere a paşilor este liniară:



Dacă unitatea de timp folosită este suficient de mare, putem scrie totul într-un singur pas:

```
1. citeşte a,b ; a ←-- (a+b)/2 ; scrie a ; Stop
```

sau (spărgînd secvenţa la diverse momente) în doi sau trei paşi, ca de exemplu

```
1. citeşte a,b
2. a ←-- (a+b)/2 ; scrie a
3. Stop
```

Ca o aplicaţie a algoritmului, pentru valorile **2** şi **8**, parcurgerea lui conduce la următoarea execuţie:

1. a ←— 2 ; b ←— 8
2. a ←— (2+8)/2 deci **a ←— 5**
3. scrie 5
4. Stop

1.2 Restul împărţirii a două numere întregi

```
rest
r,a,b  întregi  /* b nenul */
1. citeşte a,b
2. r ←-- a-(a/b)*b
3. scrie r
4. Stop
```

Vom face următoarele observaţii:

- Ca şi anterior (la 1.1.), dacă variabilele a sau b nu mai sînt folosite, oricare din ele poate lua locul lui r.

■ Deoarece `a` şi `b` sînt declarate întregi, `a/b` este tot număr întreg. Pentru `a,b` reale problema împărţirii cu rest nu se pune. Ca o lacună, schema logică nu oferă această informaţie relativă la tipul variabilelor.

Dacă nu sîntem atenţi, o scriere neglijentă a pasului 2 poate genera erori deoarece în mulţimea numerelor întregi reprezentate în memoria unui calculator, **asociativitatea şi distributivitatea nu mai funcţionează.**

Astfel, `a-b*a/b` va da totdeauna **0**, pentru că `b*a` se divide exact cu `b` şi dă cîtul `a`; Variante corecte pentru pasul 2 pot fi considerate:

```
a-b*(a/b) , a-(a/b)*b , a-int(a/b)*b ,
a-a/b*b , a-b*int(a/b)
```

iar greşite:

```
a-b*a/b , a*(1-1/b*b) .
```

O execuţie cu **b** ←--- **0** duce la o blocare pe operaţia de împărţire (de tipul *"număr prea mare "*). De obicei se ia precauţia de a folosi un bloc de decizie în care să testăm dacă `b=0` (eliminînd astfel orice eventualitate nedorită).

Dacă acest algoritm este folosit ca subalgoritm în care `a` şi `b` nu se citesc ci se aduc din algoritmul principal, iar `r` este o valoare de răspuns, putem scrie:

```
rest(a,b,r)
a,b,r  întregi
1. r ←--- a-a/b*b
2. Revenire
```

De remarcat că operaţia **Stop** este înlocuită cu **Revenire**, pentru a asigura întoarcerea la algoritmul care apelează. În această situaţie,

CITIREA ŞI SCRIEREA SÎNT CONTROLATE DE ALGORITMUL PRINCIPAL, NU DE SUBALGORITM.

Să luăm ca exemplu valorile de intrare **15** şi **9**.

Pentru prima variantă (ca algoritm), se va trece prin succesiunea de paşi:

1. a ←--- 15 , b ←--- 9
2. r ←--- 15/9*9 ≡ 15-1*9 ≡ 6
3. scrie 6
4. Stop

În cazul cînd lucrăm cu subalgoritm, el va fi apelat de un algoritm principal prin `rest(15,9,r)`. Se activează atunci `rest(a,b,r)` astfel:

a ←--- 15 , b ←--- 9
1. r ←--- 15-15/9*9 ≡ 15-1*9 ≡ 6
2. Revenire

şi se revine în algoritm, în care variabila `r` are acum valoarea **6**.

O condiţie de forma `a≥b` nu este necesară;
dacă `a<b` atunci `a/b = 0` şi deci `r ←--- a`.

1.3 Permutarea a două elemente

Foarte des este necesar ca două valori, aflate în variabile distincte să fie permutate între ele. Permutarea se face folosind operaţia de atribuire. Dar, conform ei, după scrierea lui `a ←--- b`, valoarea care se afla anterior în `a` este ştearsă. Pentru a salva conţinutul lui `a`, avem nevoie de o a treia variabilă `c` şi atunci putem scrie:

```
perm_1(a,b)
a,b,c reale
1. c ←--- a
2. a ←--- b
3. b ←--- c
4. Revenire
```

Cu prima regulă, valoarea lui a este " salvată " în c. În faza a doua, este posibil ca în a să aducem valoarea lui b, iar în final, valoarea iniţială a lui a (aflată în c) se depune în b.

Să luăm un exemplu: cum se execută `perm_1(3,0)` :

	a	b	c
Iniţial avem valorile	3	0	-
cu c ←--- a se obţine	3	0	3
cu a ←--- b se obţine	0	0	3
cu b ←--- c se obţine	0	3	3

c a fost folosită doar ca variabilă auxiliară şi în final conţinutul ei nu mai interesează.

Intuitiv, totul poate fi gîndit ca o problemă de logică:

În două pahare avem vin alb şi vin roşu. Cum trebuie procedat pentru a permuta cele două soiuri de vin din pahare ? (vinul alb să-l ducem în paharul unde era vinul roşu, iar vinul roşu în locul celui alb).

Soluţia poate fi de aceea numită şi **regula celor trei pahare.**

De remarcat că efectul este acelaşi şi în cazul secvenţei

```
1. c ←--- b ; b ←--- a ; a ←--- c
```

Folosind abilităţi de calcul, putem face permutarea celor două valori aflate în a şi b **fără a fi nevoie de** c. Pentru aceasta, să studiem secvenţa următoare:

```
perm_2(a,b)
1. a ←--- a+b
2. b ←--- a-b
3. a ←--- a-b
4. Revenire
```

Iniţial avem în locaţiile lui `a` şi `b` valorile α respectiv β.

		a	b
		α	β
După pasul	`1. a <--- a+b`	$\alpha+\beta$	β
Apoi	`2. b <--- a-b`	$\alpha+\beta$	$(\alpha+\beta)-\beta \equiv \alpha$
şi, în sfîrşit	`3. a <--- a-b`	$(\alpha+\beta)-\alpha \equiv \beta$	α

O apelare `perm_2(2,5)` de exemplu trece prin secvenţa

1. a ←--- 2+5 ≡ 7
2. b ←--- 7-5 ≡ 2
3. a ←--- 7-2 ≡ 5

Avantajul unui astfel de algoritm este clar: nu se folosesc variabile auxiliare. Dezavantajul este ceva mai greu de sesizat şi este legat de **aplicarea algoritmului într-un limbaj de programare.**

Să presupunem că `a` şi `b` sînt întregi (problema este aceeaşi pentru numere reale, doar că se lucrează cu valori mult mai mari). În calculator, numerele întregi nu trebuie să depăşească 32 767 (altfel sînt necesare precauţii speciale).

Atunci o apelare `perm(32767,3)` de exemplu ar genera un rezultat fals (fără să anunţe eroare) deoarece:

a+b ≡ 32767+3 ≡ 32770, număr care în `a` este reţinut ca **-2** .

Deci , **a ←--- -2**

după care urmează **b ←--- -2-3 ≡ -5**

a ←--- -2-(-5) ≡ 3,

numere total diferite de cele solicitate pentru a fi permutate.

EXPLICAŢIE: Oricărei variabile întregi i se asigură 16 poziţii binare, din care prima codifică semnul (0 pentru numerele pozitive, 1 pentru cele negative). Astfel, cel mai mare număr pozitiv care poate fi reţinut este

0 111 1111 1111 1111 corespunzător scrierii în binar a lui 32 767.

Adunînd 3 la acest număr, calculatorul adună în binar valoarea de sus cu

0 000 0000 0000 0011 şi ajunge la 1 000 0000 0000 0010 ,

corespunzător lui -2.

DECI TREBUIE TOTDEAUNA SĂ SE FACĂ DISTINCŢIE ÎNTRE ALGORITMUL TEORETIC, ADEVĂRAT PENTRU UN CALCULATOR IDEAL, ŞI REALIZĂRILE LUI ÎN DIVERSE LIMBAJE DE PROGRAMARE, UNDE INTERVIN DIVERSE RESTRICŢII.

1.4 Calculul valorii polinomului $p(x)=ax^2+bx+c$

Se pot da doi algoritmi diferiţi pentru rezolvarea acestei probleme:

```
 valtr_1(a,b,c)
 a,b,c,x,p reale
1. citeşte x
2. p ←-- a*x+b
3. p ←-- p*x+c
4. scrie p
5. Stop
```

```
 valtr_2(a,b,c)
 a,b,c,x,p reale
1. citeşte x
2. p ←-- a*x*x+b*x+c
3. scrie x
4. Stop
```

În ambele variante am făcut distincţie între coeficienţii polinomului `a,b,c` (care, introduşi dintr-un algoritm apelant dau contextului ideea că lucrăm cu constante) şi variabila x (pentru care se calculează valoarea).

Să exemplificăm pentru trinomul **$p(x)=2x^2-x+3$** şi **x ←--- 1 .**

`valtr_1(2,-1,3)` va trece secvenţial prin:

a ←--- 2 , b ←--- -1 , c ←--- 3
1. x ←--- 1
2. p ←--- 2*1+(-1) ≡ 1
3. p ←--- 1*1+3 ≡ 4
4. scrie 4
5. Stop

```
valtr_2(2,-1,3)
```
a ←--- 2 , b ←--- -1 , c ←--- 3
1. **x ←--- 1**
2. **p ←--- 2*1*1+(-1)*1+3 ≡ 4**
3. **scrie 4**
4. **Stop**

Deci **p(1)=4**

Observaţii :

♦ Varianta `valtr_2` conţine numai 4 paşi faţă de `valtr_1`. Totuşi, prima este mai interesantă, din două motive:

i. Dă un exemplu sugestiv al modului de utilizare a operaţiei de asignare. La pasul 4 se calculează

```
p*x+c ≡ (a*x+b)*x+c ≡ a*x*x+b*x+c
```

(valoarea "veche " a lui `p` este calculată la pasul 3, valoarea "nouă" este rezultatul final).

ii. Oferă posibilitatea de generalizare pentru un polinom de gradul `n`.

♦ Varianta `valtr_1` are 2 înmulţiri şi 2 adunări;
varianta `valtr_2` - 3 înmulţiri şi 2 adunări.

♦ În `valtr_1` pasul 2 se putea scrie şi

```
p ←-- a*x^2+b*x+c
```

În general însă este de preferat să se lucreze cu operaţia de înmulţire în loc de exponenţiere deoarece se execută mai rapid şi - în plus - la realizarea practică multe limbaje de programare nici nu posedă o astfel de operaţie.

♦ O scriere `p(x)` în loc de `p` este greşită deoarece, pentru ca `p` să fie o variabilă (conform operaţiei de asignare) trebuie ca:

i. `x` să fie număr natural

ii. `p` să fie un tablou (vezi secţiunea III.3.2) de dimensiune suficient de mare în care `p(x)` va fi numai o componentă (cea cu numărul de ordine `x`).

EXERCIŢII

1. Să se construiască algoritmi care să determine:
- media geometrică a două numere;
- media aritmetică a trei numere.

2. Se citeşte un număr **a**. Să se calculeze
1. $\mathbf{a}^8$ folosind trei operaţii de înmulţire;
2. $\mathbf{a}^6$ folosind trei operaţii de înmulţire;
3. $\mathbf{a}^7$ folosind patru operaţii de înmultire.

3. Se citeşte un număr real pozitiv **a**. Să se calculeze $\sqrt[4]{a}, \sqrt[8]{a^3}$.

Se poate determina $\sqrt[8]{a^5}$ folosind numai o înmulţire ?

4. Se dau patru numere **(a,b,c,d)**. Cum se pot permuta ele ciclic în **(d,a,b,c)**
- folosind o singură variabilă suplimentară şi aplicînd de un număr minim de ori regula celor trei pahare;
- fără a folosi nici o variabilă suplimentară.

5. Se dau patru numere **(a,b,c,d)**. Să se obţină în aceleaşi variabile valorile **(d,c,b,a)**:
- folosind o variabilă suplimentară de memorie. De cîte ori se foloseşte regula celor trei pahare ?
- fără a folosi variabile suplimentare de memorie.

6. Să se dea exemple de cazuri cînd

(a+b)+c≠a+(b+c) ; **a/b+c/b≠(a+c)/b** ; **a*(b+c)≠a*b+a*c** ;

7. Fie algoritmul:

```
perm_3(a,b)
a,b reale
1. a ←--- a-b
2.. b ←--- a+b
3. a ←--- b-a
4. Revenire
```

i. Să se arate că `perm_3(a,b)` permută variabila `a` cu `b`;

ii. Atunci cînd `perm_2(a,b)` dă depăşire, `perm_3(a,b)` va da rezultatul corect;

iii. Cînd nu se poate aplica `perm_3(a,b)` ?

8. Se dă algoritmul:

```
   perm_4(a,b)
   a,b reale
1. a ←--- a/b
2. b ←---a*b
3. a ←--- b/a
4. Revenire
```

În ce condiţii asigură el permutarea conţinutului celor două variabile ?

9. Se dau `a` şi `b` întregi. Fără a folosi variabile suplimentare, să se dea un algoritm care să reţină în variabila `a` valoarea lui `b`, iar în `b` - valoarea restului împărţirii lui `a` la `b`.

10. Să se calculeze valoarea funcţiei

f(x)=(a*x+b)/(c*x+d) .

III.2 ALGORITMI CU RAMIFICAŢII

Această clasă de algoritmi se caracterizează prin două proprietăţi:

- apar operaţiile de decizie alternativă ca singure operaţii de decizie;
- fiecare pas al algoritmului este executat cel mult odată.

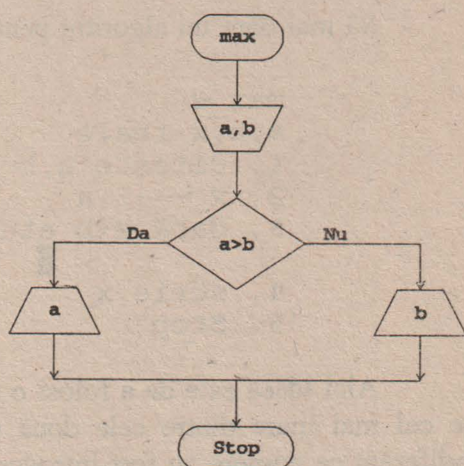
Parcurgerea unui astfel de algoritm continuă să păstreze o structură liniară dar este posibil să existe drumuri care să nu treacă prin toţi paşii algoritmului.

Dacă există o reprezentare sub formă de schemă logică, atunci vom avea doar un număr finit de drumuri de la primul la ultimul nod al schemei.

2.1 Aflarea maximului între două sau trei numere

Acesta constituie unul din cele mai simple exemple de algoritm cu ramificaţii.

```
 max_1
 a,b reale

1. citeşte a,b

2. dacă a>b atunci scrie a
            altfel scrie b

3. Stop
```



De remarcat că la pasul 2, după ce se scrie unul din numere (cel mai mare) se trece la pasul 3 - **Stop**. Reprezentarea

```
2. dacă a>b atunci scrie a
                ■
3. scrie b
4. Stop
```

este greşită: în cazul cînd a este cel mai mare, după scrierea sa la pasul 2, se trece la pasul 3 unde se scrie şi b (deci apar scrise ambele numere).
Pentru a corecta această greşeală, este necesar să completăm pasul 2 astfel:

```
2'. dacă a>b atunci scrie a;
                    salt la 4.
                 ■
```

unde `salt la 4` semnifică trecerea la pasul 4 al algoritmului, după ce s-a scris a.

În caz că ambele numere sînt egale, se scrie b (egal cu a); la o testare a≥b s-ar scrie a, variantă la fel de corectă.

Să mai dăm un algoritm pentru aceeaşi problemă:

```
max_2
a,b,x reale
1. citeşte a,b
2. x ←--- a
3. dacă x<b atunci x ←--- b
                ■
4. scrie x
5. Stop
```

Aici ideea este de a folosi o altă variabilă reală - x, în care să aducem pe cel mai mare dintre cele două numere. Scrierea se face doar pentru x, indiferent ce numere au fost introduse.

Evident, varianta max_1 prezintă o serie de avantaje în comparaţie cu max_2:

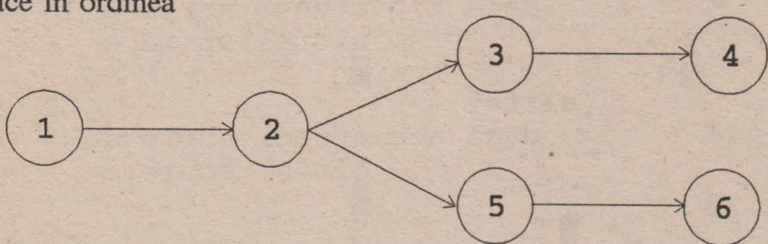
	max_1	max_2
număr paşi	3	5
total variabile	2	3
operaţii asignare	-	2
operaţii decizie	1	1
operaţii ieşire	2	1

Varianta max_2 se bucură însă de un atu pe care îl vom exploata ulterior: poate fi generalizată la un număr arbitrar de variabile.

Pentru 3 numere, să folosim algoritmul max_1 în care vom adăuga şi variabila c. Se obţine:

```
max_3
a,b,c reale
1. citeşte a,b,c
2. dacă a<b atunci salt la 5
            █
3. dacă a>c atunci scrie a
            altfel scrie c
            █
4. Stop
5. dacă b>c atunci scrie b
            altfel scrie c
            █
6. Stop
```

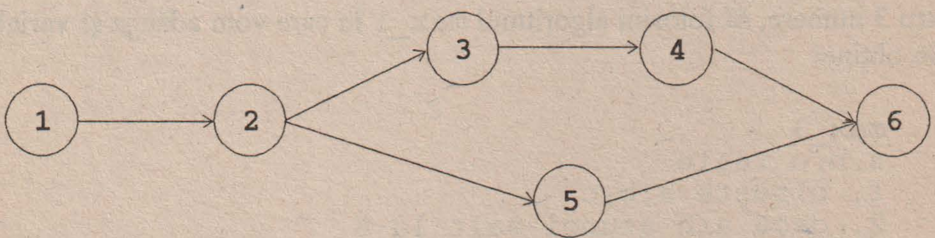
Deci, numărul de paşi ai algoritmului s-a dublat. Trecerea de la unul la altul se face în ordinea



structură de parcurgere care justifică numele de algoritm cu ramificaţii. Se poate înlocui pasul 4 cu

```
4'. salt la 6
```

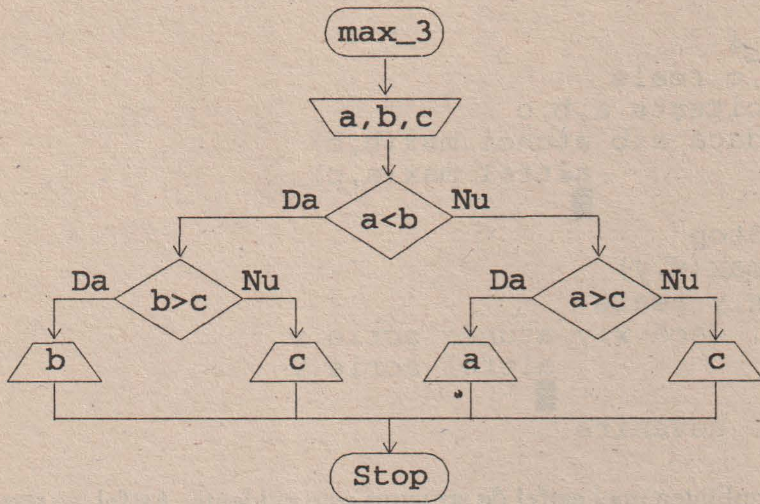
ceea ce conduce la o singură operaţie **Stop** şi la structura



Numărul de paşi se poate reduce printr-o scriere mai eficientă, în care vom elimina operaţiile de salt fără condiţie cum ar fi:

```
1. citeşte a,b,c
2. dacă a<b atunci
             dacă b>c atunci scrie b
                      altfel scrie c
                      ■
           altfel
             dacă a>c atunci scrie a
                      altfel scrie c
                      ■
           ■
3. Stop
```

Schema logică pentru algoritmul max_3 este:



Să verificăm algoritmul max_3 pentru două seturi de valori numerice:

i. (2 , 0 , -5)

1. **a ←--- 2 , b ←--- 0 , c ←--- -5**

2. Testul a<b are valoarea logică **fals** (**2 < 0 ≡ fals**). În acest caz, conform operaţiei de decizie alternativă forma scurtă, se trece la pasul 3.

3. Testul a>c are valoarea logică **adevărat** (**2 > -5 ≡ adevărat**), ceea ce conduce la scrie a.

Într-adevăr, în a se află cel mai mare dintre cele 3 numere: **2.**

ii. (0 , 8 , 1)

1. **a ←--- 0 , b ←--- 8 , c ←--- 1**

2. **(a < b) ≡ (0 < 8) ≡ adevărat** ⇒ se trece la pasul 5

5. **(b > c) ≡ (8 > 1) ≡ adevărat** şi se scrie b, unde se află valoarea **8.**

Propunem să se verifice că ajungem la acelaşi rezultat **indiferent** în care variabile introducem valorile 0, 1, 8.

Similaritatea de calcul între paşii 3.-5. şi 4.-6. conduce la ideea de a folosi un subalgoritm.

Obţinem astfel:

```
max_4
a,b,c reale
1. citeşte a,b,c
2. dacă a<b atunci max(b,c)
            altfel max(a,c)
            ■
3. Stop
   max(x,y)
   x,y reale
  1. dacă x>y atunci scrie x
              altfel scrie y
              ■
  2. Revenire
```

Simplitatea unei astfel de structuri este evidentă. Astfel, se testează dacă `a<b`; în caz afirmativ se trece la execuţia algoritmului `max` cu asignările
`x ←--- b , y ←--- c`.

Altfel, subalgoritmul `max` este activat avînd valorile
`x ←--- a , y ←--- b .`

2.2 Testarea dacă două numere întregi se divid unul cu altul

Deoarece în general este greu de urmărit ordinea de introducere a celor două numere, în algoritm se va testa dacă numărul mai mare se divide cu cel mai mic.

```
     div_1
     a,b,c întregi
    1. citeşte a,b
    2. dacă a≥b atunci salt la 4
                     █
    3. c ←--- a ; a ←---b ; b ←--- c
    4. c ←--- a-a/b*b
    5. dacă c=0 atunci scrie a' este divizibil cu ',b
               altfel scrie a' nu este divizibil cu ',b
                     █
    6. Stop
```

De reţinut:

✧ Algoritmul tratează o proprietate a numerelor întregi; deci toate variabilele `a,b,c` ale algoritmului vor fi considerate ca atare;

✧ Pasul 3 este de fapt "regula celor 3 pahare " folosită pentru a aduce în `a` pe cel mai mare dintre cele două numere, iar în `b` - pe cel mai mic. În acest fel, chiar dacă numerele au fost introduse invers, rezultatul va fi cel corect.

Acelaşi lucru îl obţinem dacă scriem pasul 3 astfel:

`3'. c ←--- b-b/a*a ; salt la 5` (practic, s-a repetat 4).

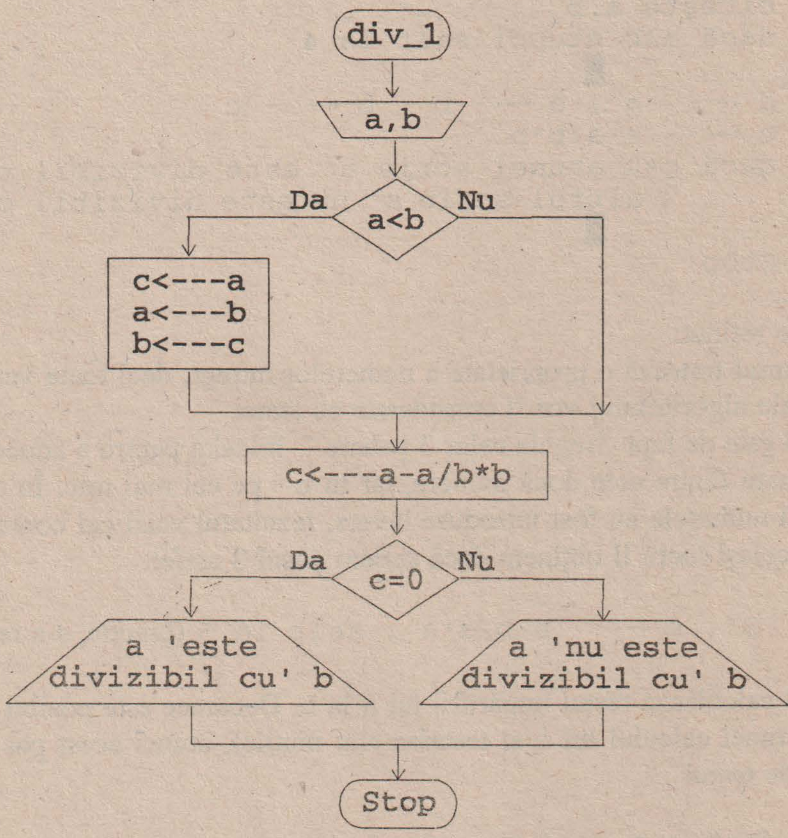
✧ Pasul 4 calculează restul împărţirii lui `a` la `b`. Deoarece este posibil să avem `a=b` (şi atunci calculul lui `c` şi testarea sînt inutile), atunci acest pas poate fi precedat de testul

```
    dacă a=b atunci salt la 6
                     █
```

✧ De remarcat utilizarea variabilei interne c. La pasul 3 c este variabilă de lucru pentru inversarea numerelor a şi b, după care conţinutul ei nu mai este necesar; de aceea la pasul 4 se depune aici restul împărţirii lui a la b, rest testat apoi prin comparare cu zero.

✧ În cele două operaţii de scriere, în afara variabilelor de ieşire a fost inserat un text. Scris între semnele ' ', el va completa informaţia solicitată de utilizator.

Să prezentăm şi schema logică corespunzătoare acestui algoritm :



✧ Verificări:

i. **a ←--- 8 , b ←--- 3**; se trece prin:
2. **(a ≥ b) ≡ (8 ≥ 3) ≡ adevărat** ⇒ 4
4. **c ←--- 8-8/3*3 ≡ 8-2*3 ≡ 2**
5. **(c = 0) ≡ (2 = 0) ≡ fals**
8 nu este divizibil cu 3 .

ii. **a ←--- 5 , b ←--- 60 :**
2. **(a ≥ b) ≡ (5 ≥ 60) ≡ fals** ⇒ 3
3. " regula celor trei pahare" conduce la **a ←--- 60 , b ←--- 5**
4. **c ←--- 60-60/5*5 ≡ 60-12*5 ≡ 0**
5. **(c = 0) ≡ (0 = 0) ≡ adevărat**
60 este divizibil cu 5

Algoritmul poate fi comasat în mai puţini paşi astfel:

```
    div_2
    a,b,c întregi
1.  citeşte a,b
2.  dacă a<b atunci c ←--- b; b ←--- a; a ←--- c
              ▮
3.  dacă a=a/b*b atunci scrie a' este divizibil cu 'b
                    altfel scrie a' nu este divizibil cu 'b
                         ▮
4.  Stop
```

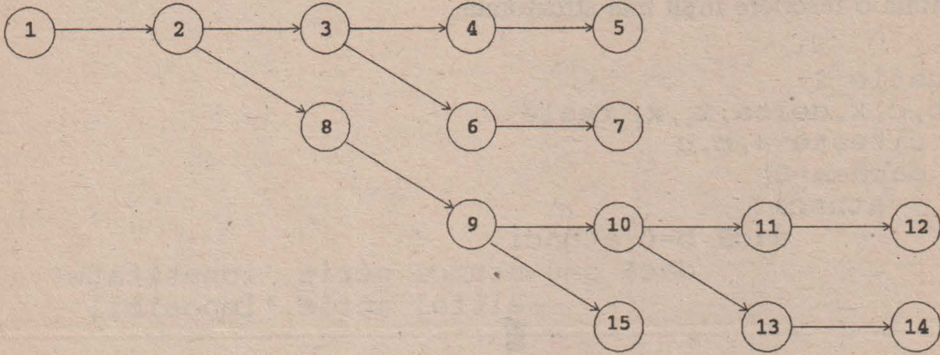
2.3 Rezolvarea ecuaţiei de grad cel mult 2

Este exemplul tipic de algoritm cu ramificaţii, în special datorită condiţiilor impuse de existenţa soluţiilor ecuaţiei $ax^2+bx+c=0$.

```
ecuaţie_1
a,b,c,x,delta,x1,x2 reale
1. citeşte a,b,c
2. dacă a≠0 atunci salt la 8

3. dacă b≠0 atunci salt la 6

4. dacă c=0 atunci scrie 'identitate'
            altfel scrie 'ecuaţie imposibilă'

5. Stop
6. x ←--- -c/b
7. scrie 'ecuaţie de gr.I cu soluţia ',x; Stop
8. delta ←--- b*b-4*a*c
9. dacă delta<0 atunci salt la 15

10. dacă delta=0 atunci salt la 13

11. x1 ←--- (-b-rădăcina(delta))/2/a;
    x2 ←--- (-b+rădăcina(delta))/2/a
12. scrie x1,x2 ; Stop
13. x ←--- -b/2/a
14. scrie 'soluţie dublă ',x; Stop
15. scrie 'ecuaţia nu are rădăcini reale '; Stop
```

Se observă că rezolvarea efectivă a ecuaţiei de gradul doi - cu discuţia aferentă - cuprinde numai jumătate din paşii algoritmului (8-15). În rest se elimină pe rînd diverse cazuri particulare generate de valorile posibile pe care le pot lua coeficienţii a, b, c.

Structura algoritmului este :



Să cercetăm cum lucrează algoritmul pentru două seturi de date:

i. Ecuaţia **x+2=0**; deci **a ←--- 0 , b ←--- 1 , c ←--- 2;**

La pasul 2, condiţia `a≠0` are valoarea logică **fals**, deci se continuă cu pasul 3. Aici `b≠0` are valoarea logică **adevărat**, ceea ce determină salt la pasul 6.

Se calculează **x ←--- -2/1 ≡ -2** după care se scrie:

ecuaţie de gradul I avînd soluţia -2.

ii. Ecuaţia **x^2-4x+2=0**; deci **a ←--- 1 , b ←--- -4 , c ←--- 2**

2. **(a ≠ 0) ≡ (1 ≠ 0) ≡ adevărat** ⇒ 8
8. **delta ←--- 16-4*1*2 ≡ 8**
9. **(delta < 0) ≡ (8 < 0) ≡ fals** ⇒ 10
10. **(delta = 0) ≡ (8 = 0) ≡ fals** ⇒ 11
11. **x_1 ←--- (4-rădăcina(8))/2/1 ≡ 0.58578644**
    x_2 ←--- (4+rădăcina(8))/2/1 ≡ 3.4142136

Paşii algoritmului `ecuaţie_1` pot fi rearanjaţi astfel încît să fie posibilă o rescriere mult mai structurată.

```
ecuatie_2
a,b,c,x,delta,x1,x2 reale
1. citeşte a,b,c
2. dacă a=0
      atunci
            dacă b=0 atunci
                  dacă c=0 atunci scrie 'identitate'
                           altfel scrie 'imposibil'
                           ■
                  altfel x ←--- -c/b ;
                           scrie 'ec.gr.I ; x=',x
                  ■
      altfel delta ←--- b*b-4*a*c ;
            dacă delta<0 atunci scrie'fără rădăcini'
                         altfel dacă delta=0 atunci
                                             x ←--- -b/2/a ;
                                         scrie'sol.unică',x
                                                altfel
                  x1 ←--- (-b-rădăcina(delta))/2/a;
                  x2 ←--- (-b+rădăcina(delta))/2/a;
                              scrie x1,x2
                                                ■
                         ■
      ■
3. Stop
```

Recomandăm să se insiste asupra acestei modalităţi de scriere şi să se parcurgă următorul algoritm:

Test:

1. Încercaţi funcţionarea versiunii structurate pentru cele două seturi de valori date anterior;
2. Introduceţi şi alte seturi de date astfel încît să se treacă prin toate cazurile posibile;
3. Refaceţi algoritmul **ecuaţie** plecînd de la acest algoritm structurat;
4. Dacă nu aţi reuşit, salt la pasul 2 altfel Stop;

2.4 Algoritm cu ramificaţii pentru o problemă de căutare

Algoritmii matematici cu ramificaţii rezolvă în general probleme cu grad de complexitate scăzut. Utilizarea lor este preponderentă în algoritmii nematematici, în special probleme de logică.

O astfel de clasă de algoritmi sînt problemele de căutare.

Să rezolvăm de exemplu problema următoare:

Fie o mulţime finită A şi doi interlocutori x şi y. x alege un obiect din A iar y caută să ghicească prin întrebări ce obiect a fost ales. Regulile jocului sînt în general următoarele:

i) x nu răspunde la întrebări decît prin "da" sau "nu"

ii) numărul de întrebări puse de y trebuie să fie cît mai mic posibil.

La un algoritm bine scris, o astfel de căutare binară (sînt permise numai două răspunsuri) permite aflarea prin k întrebări a unui obiect dintr-o mulţime **A** avînd 2^k elemente.

Să particularizăm problema pentru lista:

A:	1. Raţă	5. Găină	9. Erete	13. Porumbel
	2. Colibri	6. Condor	10. Pinguin	14. Struţ
	3. Cîine	7. Cal	11. Antilopă	15. Leu
	4. Rîmă	8. Fluture	12. Şarpe	16. Greiere

```
    joc_1
    x şir de caractere
 1. scrie 'Alegeţi un nume de vietate din lista A. El
    poate fi ghicit din patru întrebări'; citeşte x
 2. dacă x 'este pasăre' atunci salt la 3
                            altfel salt la 10
```

```
3. dacă x 'trăieşte la noi în ţară' atunci salt la 4
                                    altfel salt la 7
                                    ■
4. dacă x 'este domestică' atunci salt la 5
                           altfel salt la 6
                           ■
5. dacă x 'îi place apa' atunci scrie 'Raţă'
                         altfel scrie 'Găină'
                         ■ ; Stop
6. dacă x 'este răpitor' atunci scrie 'Erete'
                         altfel scrie 'Porumbel'
                         ■ ; Stop
7. dacă x 'zboară' atunci salt la 8
                   altfel salt la 9
                   ■
8. dacă x 'este mică' atunci scrie 'Colibri'
                      altfel scrie 'Condor'
                      ■ ; Stop
9. dacă x 'trăieşte la Pol' atunci scrie 'Pinguin'
                            altfel scrie 'Struţ'
                            ■ ; Stop
10. dacă x 'este animal' atunci salt la 11
                         altfel salt la 14
                         ■
11. dacă x 'este domestic' atunci salt la 12
                           altfel salt la 13
                           ■
12. dacă x 'latră' atunci scrie 'Cîine'
                   altfel scrie 'Cal'
                   ■ ; Stop
13. dacă x 'este ierbivor' atunci scrie 'Antilopă'
                           altfel scrie 'Leu'
                           ■ ; Stop
14. dacă x 'se tîrăşte' atunci salt la 15
                        altfel salt la 16
                        ■
15. dacă x 'muşcă' atunci scrie 'Şarpe'
                   altfel scrie 'Rîmă'
                   ■ ; Stop
16. dacă x 'cîntă' atunci scrie 'Greiere'
```

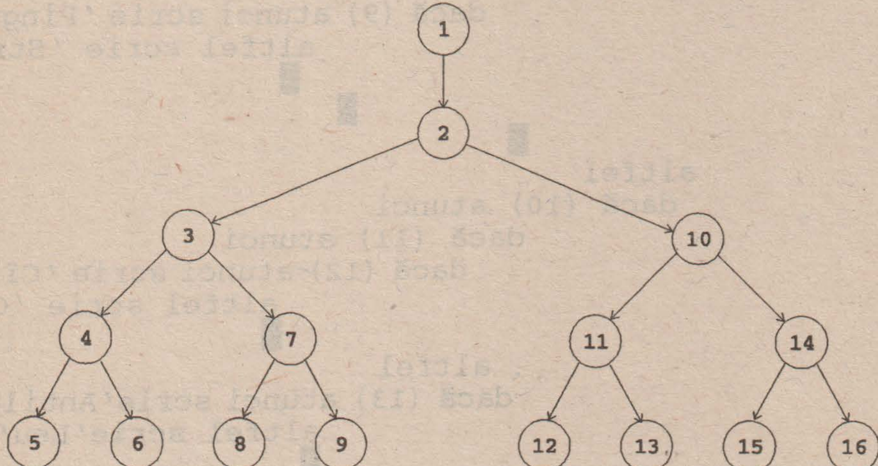
```
          altfel scrie 'Fluture'
          █ ; Stop
```

Observaţii:

♦ x nu este un număr ci un cuvînt (un şir de caractere); testele nu mai sînt exprimate prin expresii relaţionale, ci prin întrebări la care sînt posibile numai răspunsuri dicotomice (**adevărat** sau **fals**).

♦ algoritmul a fost construit omogen, astfel încît pentru orice obiect să fie necesare patru întrebări. Este posibilă şi construcţia în care unele obiecte să fie găsite prin mai puţin (sau mai mult) de patru întrebări - în funcţie de şansă. Din multe puncte de vedere însă, **o astfel de căutare nu este optimă.** În cazul particular al acestei probleme, tot prin 15 (maxim) teste putem întreba pe rînd: `x`'este raţă ?', `x`'este găină ?', etc. Pentru situaţia cea mai nefavorabilă, elementul căutat este aflat după 15 întrebări (algoritm liniar), pe cînd în `joc1`, sînt necesare numai 4.

♦ structura algoritmului `joc1` este greu de urmărit. Schema de parcurgere a paşilor săi este intuitiv mai clară şi constituie oarecum o justificare a termenului de "căutare binară":



Scris structurat, algoritmul se prezintă astfel:

```
joc_1
 1. citeşte x
 2. dacă (2) atunci
          dacă (3) atunci
                  dacă (4) atunci
                       dacă (5) atunci scrie 'Raţă'
                                altfel scrie 'Găină'
                                ■
                           altfel
                       dacă (6) atunci scrie 'Erete'
                                altfel scrie 'Porumbel'
                                ■
                           ■
                      altfel
                   dacă (7) atunci
                       dacă (8) atunci scrie 'Colibri'
                                altfel scrie 'Condor'
                                  ■
                               altfel
                        dacă (9) atunci scrie 'Pinguin'
                                 altfel scrie 'Struţ'
                                  ■
                               ■
                      ■
              altfel
               dacă (10) atunci
                      dacă (11) atunci
                         dacă (12) atunci scrie 'Cîine'
                                   altfel scrie 'Cal'
                                   ■
                        altfel
                       dacă (13) atunci scrie'Antilopă'
                                 altfel scrie'Leu'
                                 ■
                        ■
```

```
                         altfel
                       dacă (14) atunci
                      dacă (15) atunci scrie'Şarpe'
                                altfel scrie'Rîmă'
                                ■
                                 altfel
                      dacă (16) atunci scrie'Greiere'
                               altfel scrie'Fluture'
                                ■
                                 ■
                           ■
              ■
3. Stop
```

Nu există unicitate în ceea ce priveşte forma întrebărilor. Din structura arborescentă se observă că este necesar ca după fiecare întrebare, răspunsul **să reducă la jumătate mulţimea alegerilor posibile.**

În acest fel, după k întrebări, o mulţime care iniţial are 2^k elemente, se reduce la o mulţime formată dintr-un singur element.

Revenind la problema anterioară, elementul căutat poate fi determinat unic şi în felul următor:

Fie listele:

Lista 1	**Lista 2**	**Lista 3**	**Lista 4**
1. antilopă	1. cal	1. antilopă	1. antilopă
2. erete	2. condor	2. cal	2. cal
3. fluture	3. găină	3. cîine	3. cîine
4. leu	4. leu	4. colibri	4. erete
5. pinguin	5. porumbel	5. condor	5. găină
6. porumbel	6. rîmă	6. leu	6. leu
7. şarpe	7. şarpe	7. pinguin	7. porumbel
8. struţ	8. struţ	8. struţ	8. raţă

```
joc_2
x şir de caractere
i întreg
1. citeşte x
2. dacă x'este în lista 1' atunci i ←--- 1
                           altfel i ←--- 0
                           ■
3. dacă x'este în lista 2' atunci i ←--- 2*i+1
                           altfel i ←--- 2*i
                           ■
4. dacă x'este în lista 3' atunci i ←--- 2*i+1
                           altfel i ←--- 2*i
                           ■
5. dacă x'este în lista 4' atunci i ←--- 2*i+1
                           altfel i ←--- 2*i
                           ■
6. dacă i>0 atunci scrie
      'Numele se află în lista A pe poziţia'i
          altfel scrie 'Greiere'
          ■
7. Stop
```

Ideea este bazată pe reprezentarea în binar a numerelor din intervalul $[0,2^4-1]$. Fiecare test de apartenenţă la o listă are ca răspuns

Da - codificat cu **1**

Nu - codificat cu **0**.

Cele patru întrebări conduc la un număr format din 4 cifre binare. Calculul aritmetic pentru `i` duce la valoarea zecimală a acestui număr, valoare zecimală care va da poziţia numelui în lista **A**.

De exemplu, un nume care se află doar în listele 2 şi 4 conduce la numărul binar 0101. Transformat în baza zece, acest număr este 5; pe poziţia 5 în lista **A** se află `"Găină"`. Într-adevăr, acest nume apare în listele 2 şi 4 şi numai aici.

Singurul caz special este 2^4=16. Reprezentarea sa în binar este 10000, în care ultimele patru cifre sînt 0000. Numele respectiv (aici `"Greiere"`) nu se află în nici o listă, deci se va obţine **i** ←--- **0**; poziţia 0 în lista **A** va corespunde numărului 16.

Evident, un astfel de algoritm nu este atît de atractiv din punct de vedere al întrebărilor, dar poate fi folosit **pentru orice listă de 2^k nume distincte indiferent de semnificaţia lor**.

Scrierea algoritmului `joc2` se poate simplifica dacă folosim subalgoritmi:

```
    joc_3
    1. citeşte x; i ←--- 0
    2. test (x,1,i)
    3. test (x,2,i)
    4. test (x,3,i)
    5. test (x,4,i)
    6. dacă i=0 atunci i ←--- 16
                  ■
    7. scrie 'Numele se află în lista A pe poziţia'i
    8. Stop
 test (x,k,i)
1. dacă x'este în lista'k atunci i ←--- 2*i+1
                          altfel i ←--- 2*i
                          ■
2. Revenire
```

EXERCIŢII

1. Să se ordoneze crescător (descrescător) o secvenţă de 3 numere reale.

2. Se dau 4 numere întregi. Cîte din ele sînt impare ?

3. Să se rezolve sistemul de ecuaţii liniare `a*x+b*y=c, a'*x+b'*y=c'`.

4. Se dau trei elemente ale unui triunghi (unghiuri, laturi). Să se determine celelalte trei elemente ale sale precum şi aria.

5. Se dau patru numere pozitive. Să se testeze dacă ele pot fi laturile unui patrulater. Poate fi acesta paralelogram ? Dar romb ?

6. Pe lîngă cele 4 numere citite la problema 5, să se mai introducă o variabilă de intrare cu ajutorul căreia să se poată lua decizia dacă un paralelogram este dreptunghi, sau un romb este patrat.

7. Se dă un număr de trei cifre. Să se elimine o cifră astfel încît numărul de două cifre rămas să fie minim posibil.

8. Se dă un număr întreg. Cîte cifre pare intră în componenţa lui ?

9. Se dă un interval `[a,b]` cu `b-a=k` şi $\alpha \in$ `[a,b]`. Să se determine un interval `[x,y]` cu proprietăţile: $\alpha \in$ `[x,y]`, `y-x=k/4`.
Ce completare trebuie făcută pentru ca `y-x=k/8` ?

10. Să se rescrie algoritmul `joc2` (`joc_3`) pentru o listă **A** formată din
i) 8 culori , ii) 16 ţări , iii) 32 nume de persoane.

11. Se dă un număr ales la întîmplare între 1 şi 500. Să se construiască un algoritm care să-l ghicească. Care este numărul minim de întrebări ?

III.3 ALGORITMI CICLICI

În marea majoritate a algoritmilor apare necesitatea ca aceşti paşi să fie parcurşi de mai multe ori. În acest fel se formează un **ciclu** (sau **buclă**).

Numărul de repetări (treceri) ale ciclului poate fi fix sau variabil.

Un ciclu este definit complet prin 3 elemente:

i. **Iniţializarea (I)**: variabilele care intră în ciclu capătă valori cu care vor începe prima parcurgere.

ii. **Condiţia de rămînere în ciclu (C)**: La fiecare parcurgere se verifică condiţia iniţială **C** (o expresie logică); dacă ea are valoarea **adevărat**, ciclul se reia; în caz contrar se iese din ciclu, trecîndu-se la următorul pas de algoritm.

Scrierea condiţiei **C** trebuie făcută cu mare atenţie; altfel este posibil ca ea să nu permită niciodată părăsirea ciclului şi atunci nu mai avem un algoritm (nu mai este îndeplinită condiţia de finitudine).

În unele situaţii este mai convenabil să testăm condiţia **nu C**; aceasta va fi **condiţia de părăsire a ciclului**. Bucla este reluată cît timp **nu C** are valoarea **fals**.

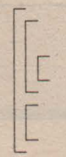
iii. **Corpul buclei (B)**: este format din grupul de paşi de algoritm care formează ciclul. Condiţii:

- **B** conţine cel puţin un pas de algoritm;
- În timpul parcurgerii lui **B** trebuiesc modificate elementele care intră în condiţia **C**; altfel **C** ia mereu aceeaşi valoare şi bucla nu este părăsită.
- În **B** pot fi şi alte cicluri.

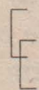
Astfel, două bucle pot fi:

a) distincte: nu au nici un pas comun:

b) incluse una în alta :

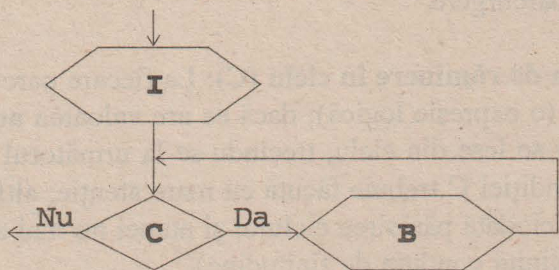


De remarcat că nu este permisă o utilizare a buclelor sub forma



Există două variante de aşezare a componentelor **I, B, C** ale unei bucle:

1.

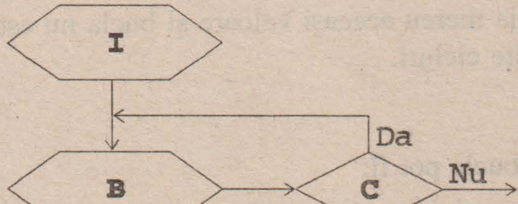


Această formă de schemă logică corespunde operaţiei de algoritm

```
I
cît timp C
execută
B
█
```

Se observă că aici este posibil - ca un caz limită - să nu fie nici o parcurgere a corpului **B** al buclei (dacă **C** este **fals** chiar de la prima testare).

2.

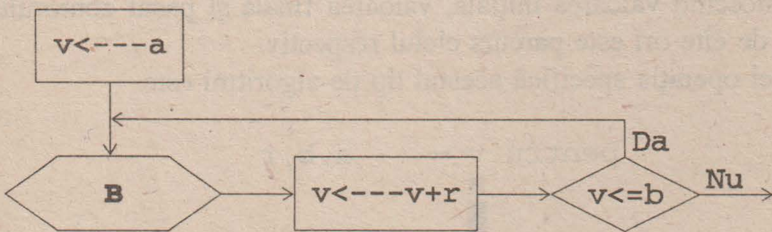


sau, tradus în operaţie de algoritm,

```
        I
     repetă
        B
pînă cînd (nu C)
```

Aici, corpul **B** este parcurs de algoritm cel puţin odată.

Ca un caz particular al acestei variante, există posibilitatea



care este folosit atît de des încît este definit cu o operaţie specială

```
pentru  v ←-- a,b,r
        B
        █
```

respectiv

```
pentru v ←-- a,b
       B
       █
```

dacă `r` are valoarea 1.

3.1 ALGORITMI CICLICI CU NUMĂR CUNOSCUT DE PAŞI

Sînt algoritmi ciclici în care numărul de treceri prin fiecare ciclu este fix. Mai sînt numiţi şi **algoritmi ciclici cu contor**, deoarece - ca o caracteristică a lor - fiecare buclă are asociată o variabilă de numărare cu scopul de a contabiliza trecerile prin ciclu.

Cunoscînd valoarea iniţială, valoarea finală şi pasul contorului, putem determina de cîte ori este parcurs ciclul respectiv.

Deci operaţia specifică acestui tip de algoritmi este

```
pentru v ←--- a,b,r
          B
          █
```

unde variabila **v** joacă rol de contor. Numărul de treceri prin corpul **B** al buclei este

max { 0, int((b-a)/r) } +1 (*) .

3.1.1 Suma numerelor naturale pînă la n

```
suma_1
x,i,n  întregi
1. citeşte n
2. x ←--- 0
3. pentru i ←--- 0,n
          x ←--- x+i
          █
4. scrie x
5. Stop
```

Înainte de a studia acest algoritm, să facem o verificare particulară.

Fie de exemplu cazul cînd **n ←--- 4.**

Primii doi paşi asigură asignările **n ←--- 4** şi **x ←--- 0 .**

Pasul 3 se execută pentru toate valorile pe care le ia `i` de la 0 la 4:

pas 3: **i ←--- 0 , x ←--- x+i ≡ 0+0 ≡ 0.**

După prima trecere prin buclă de la pasul 3 avem **i ←--- 0, x ←--- 0.**
Urmează **i ←--- i+1**; testul **(i ≤ n) ≡ (1 ≤ 4) ≡ adevărat** asigură reluarea pasului 3 cu **i ←--- 1, x ←--- 0.** Avem

pas 3: **x ←--- x+i ≡ 0+1 ≡ 1**
i ←--- i+1 ≡ 1+1 ≡ 2
(i ≤ n) ≡ (2 ≤ 4) ≡ adevărat şi pasul 3 se reia.

pas 3: **x ←--- x+i ≡ 1+2 ≡ 3**
i ←--- i+1 ≡ 2+1 ≡ 3
(i ≤ n) ≡ (3 ≤ 4) ≡ adevărat

deci se revine la pasul 3 avînd asignările **i ←--- 3, x ←--- 3.**

pas 3: **x ←--- x+i ≡ 3+3 ≡ 6**
i ←--- i+1 ≡ 3+1 ≡ 4
(i ≤ n) ≡ (4 ≤ 4) ≡ adevărat

din nou, pasul 3 este reluat, cu **i ←--- 4** şi **x ←--- 6.**

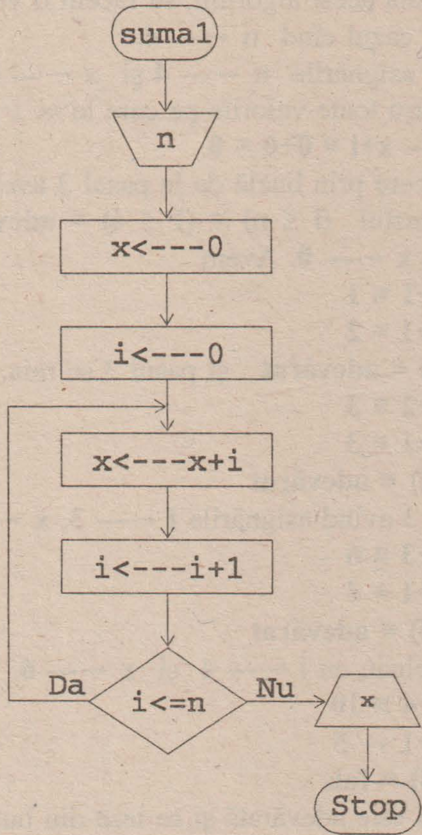
pas 3: **x ←--- x+i ≡ 6+4 ≡ 10**
i ←--- i+1 ≡ 4+1 ← 5
(i ≤ n) ≡ (5 ≤ 4) ≡ fals

Condiţia nu mai este adevărată şi se iese din bucla de la pasul 3 cu
x ←--- 10.

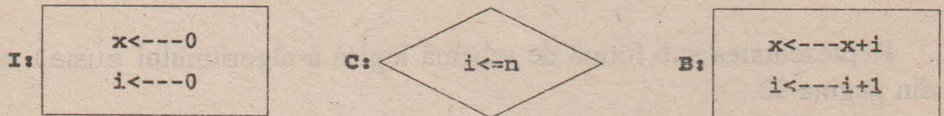
pas 4: **scrie 10**
Stop

Într-adevăr , 0+1+2+3+4=10.

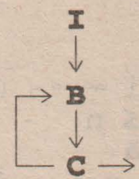
Reprezentarea sub formă de schemă logică a algoritmului `suma1` este cea din pagina 92.



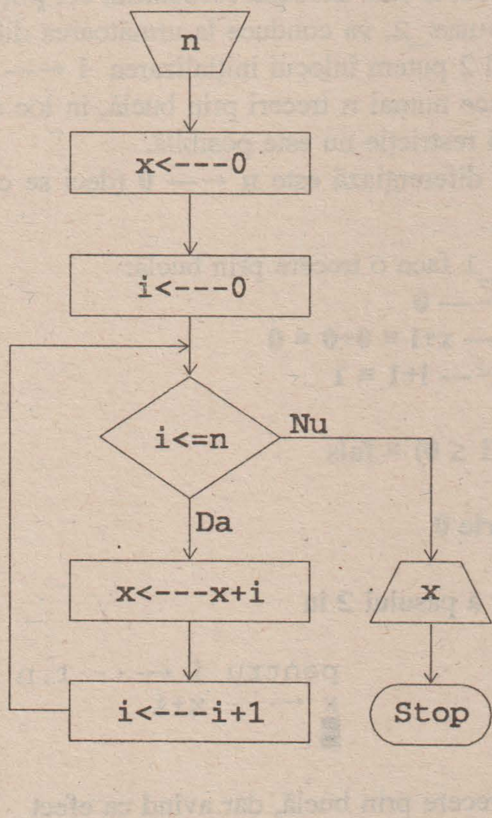
Cele trei blocuri caracteristice algoritmilor ciclici sînt:



şi ele sînt aşezate în ordinea



O ordonare inversă între blocurile **B** şi **C** conduce la schema logică



Algoritmul scris pe baza acestei forme este :

```
suma_2
x,i,n  întregi
1. citeşte n
2. x ←--- 0 ; i ←--- 0
3. cît timp i ≤ n
      execută
      x ←--- x+i ; i ←--- i+1
      ▮
4. scrie x
5. Stop
```

Faptul că blocul **B** este executat **totdeauna cel puţin odată**, ceea ce nu este valabil pentru `suma_2`, va conduce la următoarea diferenţiere de scriere: În `suma_2`, la pasul 2 putem înlocui iniţializarea **i ←--- 0** cu **i ←--- 1.** Astfel, aici se vor face numai `n` treceri prin buclă, în loc de `n+1`.
La `suma_1`, această restricţie nu este posibilă.
Cazul limită care le diferenţiază este **n ←--- 0** (deci se cere suma numerelor naturale pînă la 0).

Aici, `suma_1` face o trecere prin buclă:

i ←--- 0
x --- x+i ≡ 0+0 ≡ 0
i ←--- i+1 ≡ 1

după care testarea

(i ≤ n) ≡ (1 ≤ 0) ≡ fals

va conduce la

scrie 0

O modificare a pasului 2 în

```
pentru i ←--- 1,n
x ←--- x+i
▮
```

ar da tot o singură trecere prin buclă, dar avînd ca efect

x ←--- x+i ≡ 1

şi rezultatul ar fi **1**, fals.

Algoritmul `suma_2` în care modificăm pasul 2 în

2'. x ←--- 0 ; i ←--- 1

va începe pasul 3 cu testarea

(i≤n) ≡ (1≤0) ≡ fals,

rezultat care închide intrarea în buclă.

Deci pasul 3 nu ar fi executat niciodată şi s-ar trece la pasul 4:

4. scrie 0 .

Dacă `n≥1`, modificarea de la `suma1` nu va afecta rezultatul. De aceea putem reface primul algoritm astfel:

```
suma_3
x,i,n  întregi
1. citeşte n
2. x ←--- 0
3. dacă n=0 atunci salt la 5

4. pentru i ←--- 1,n
          x ←--- x+i

5. scrie x
6. Stop
```

Tot `suma_1` se poate prezenta sub forma

```
suma_4
x,i,n  întregi
1. citeşte n
2. x ←--- 0 ; i ←--- 0
3. repetă
        x ←--- x+i ; i ←--- i+1
        pînă cînd (i>n)
4. scrie x
5. Stop
```

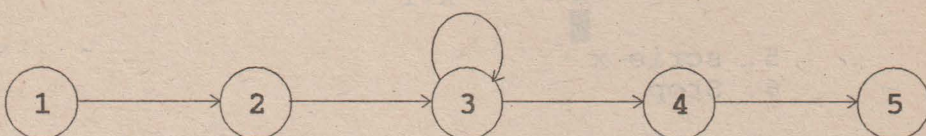
sau, dacă nu ţinem cont de operaţiile caracteristice algoritmilor ciclici,

```
suma_5
x,i,n  întregi
1. citeşte x
2. x ←-- 0 ; i ←-- 0
3. x ←-- x+i ; i ←-- i+1
4. dacă i≤n  atunci salt la 3
             altfel scrie x

5. Stop
```

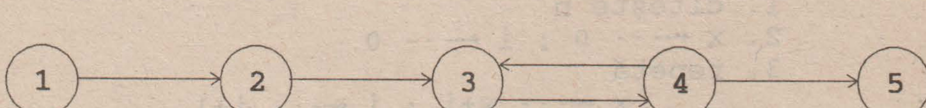
Toate aceste variante corespund unei aceleiaşi scheme logice. De aceea se consideră că

O SCHEMĂ LOGICĂ REPREZINTĂ O CLASĂ DE ALGORITMI

Structura algoritmilor suma_1, suma_2 şi suma_4 este:



iar pentru suma_5,



Variabilele folosite:

✦ **Variabila de intrare** `n`; dă numărul la care se opreşte calculul sumei. După cum am văzut la diferenţele între `suma_1` şi `suma_2` modificată, o atenţie deosebită trebuie acordată anumitor valori limită ale variabilei de intrare.

De asemenea este recomandabil ca algoritmul să conţină testări de verificare a corectitudinii datelor de intrare (de forma `n≥0`).

✦ **Variabila contor** `i`; în aceşti algoritmi ea are o valoare iniţială **i ←--- 0** (sau **i ←--- 1** la `suma2` modificată), o valoare finală **i ←--- n+1** care determină părăsirea buclei, şi un pas **1**.

De remarcat că o operaţie `pentru` include implicit în ea şi asignarea `i ←---i+1`; în cazul altor operaţii iterative (`cît timp`, `repetă`, `dacă-atunci`), această mărire a contorului trebuie scrisă efectiv.

O creştere a variabilei contor cu o valoare pozitivă se numeşte **incrementare**, iar cu o valoare negativă - **decrementare**.

De exemplu, acelaşi algoritm poate fi scris folosind decrementarea, astfel:

```
suma_6
x,i,n  întregi
1. citeşte n
2. x ←--- 0
3. pentru i ←--- n,0,-1
          x ←--- x+i
          █
4. scrie x
5. Stop
```

Aici, deoarece pasul este **-1** (şi nu **1**), el a trebuit să fie scris explicit .
În acest caz, valoarea iniţială a contorului a fost `n` iar valoarea finală **-1**.

Pentru **n ←--- 4** se obţine pe rînd

x ←--- 4, x ←--- 4+3 ≡ 7, x ←--- 7+2 ≡ 9,
x ←--- 9+1 ≡ 10, x ←--- 10+0 ≡ 10.

Şi aici, dacă încercăm să optimizăm numărul de treceri prin buclă făcînd iniţializarea x ←--- n şi plecînd cu i ←--- n-1,0,-1, algoritmul nu va funcţiona în cazul particular **n ←--- 0**.

✦ **Variabila internă** x care este utilizată şi ca **variabilă de ieşire**; constituie zona de bază pentru calcule. În ea se reţin toate sumele parţiale de numere naturale, pînă la trecerea finală prin ciclu, cînd s-a acumulat suma tuturor numerelor de la **0** la n. Un moment important este iniţializarea acestei variabile; ea este x ←--- 0 din două motive:

✧ **0** este suma minimă posibilă a primelor numere naturale;

✧ În x memorîndu-se rezultate de adunări, pentru a nu se vicia rezultatul final se pleacă de la un element care adunat iniţial cu orice număr, îl lasă nemodificat (un astfel de număr se numeşte **element neutru**). Acesta nu poate fi decît **0**.

Evident, tot blocul de calcul **B** poate fi considerat ca un subalgoritm şi prelucrat ca atare. Nu recomandăm totuşi o astfel de utilizare deoarece - practic, scoaterea variabilelor contor înafara ciclului poate crea perturbaţii (efecte secundare) nedorite la implementarea algoritmului în limbaje de programare. Deci:

```
suma_7
x,i,n  întregi
1. citeşte n
2. x ←--- 0
3. pentru i ←--- 0,n
          suma(x,i)
          █
4. scrie x
5. Stop
  suma(a,b)
  a,b  întregi
 1. a ←--- a+b
 2. Revenire
```

este corectă, dar nu şi recomandabilă.

3.1.2 Cîte elemente dintr-un şir sînt mai mari decît o valoare dată

Avem de rezolvat următoarea problemă:

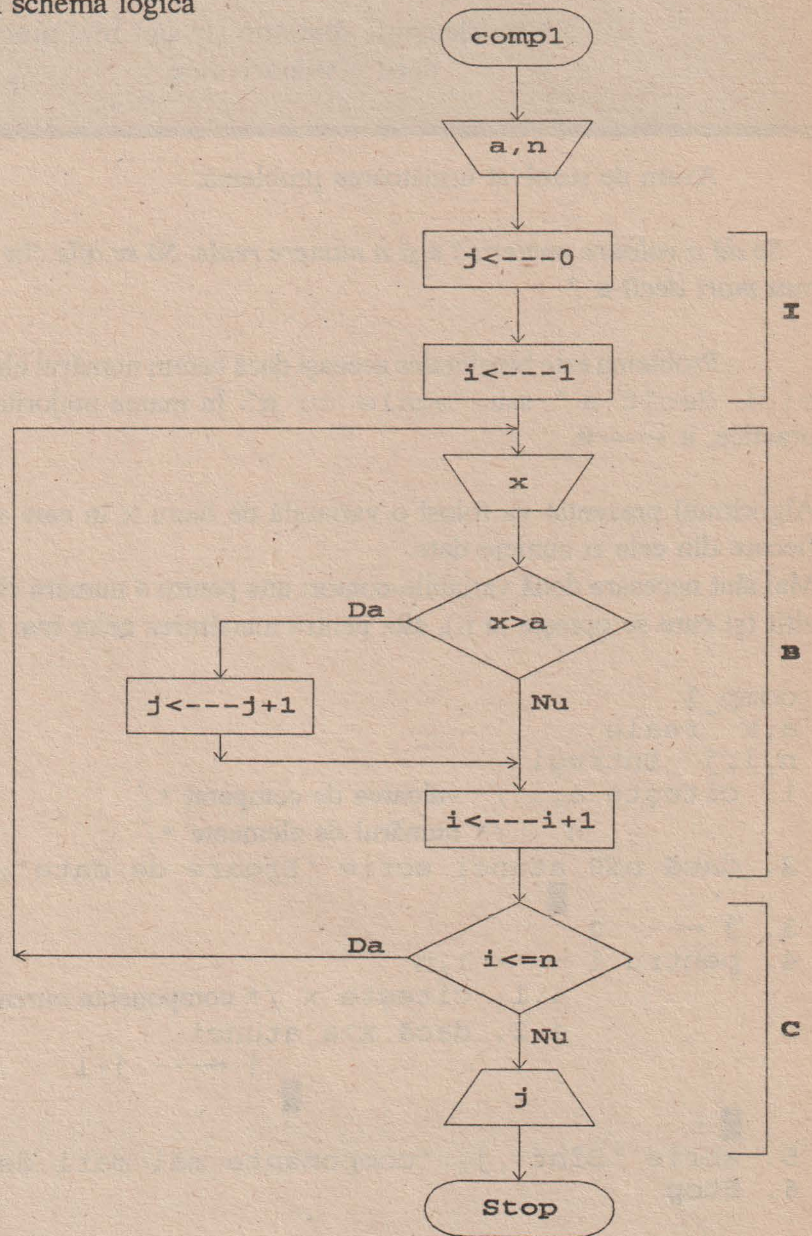
*Se dă o valoare numerică **a** şi **n** numere reale. Să se afle cîte din acestea sînt mai mari decît **a** ?*

Problema este bineînţeles aceeaşi dacă cerem numărul elementelor "`mai mici decît` **a**" sau "`egale cu` **a**". În marea majoritate a aplicaţiilor practice, **a** ←--- **0**.

Algoritmul prezentat va folosi o variabilă de lucru `x` în care vom citi pe rînd fiecare din cele `n` numere date.
Mai sînt necesare două variabile-contor: una pentru a număra cîte variabile s-au citit (şi care se opreşte la `n`), alta pentru numărarea celor mai mari decît `a`.

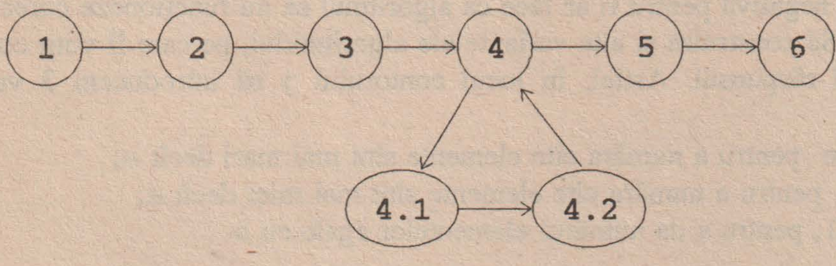
```
comp_1
a,x  reale
n,i,j  întregi
1. citeşte a,  /* valoarea de comparat */
           n   /* numărul de elemente */
2. dacă n≤0 atunci scrie 'Eroare de date'; Stop
           █
3. j ←--- 0
4. pentru i ←--- 1,n
           4.1. citeşte x /* componenta curentă */
           4.2. dacă x>a atunci
                           j ←--- j+1
                        █
   █
5. scrie 'Sînt',j, 'componente mai mari decît ',a
6. Stop
```

caracterizat prin schema logică



După cum se observă, operaţia de citire nu este obligatoriu pusă la începutul algoritmului; astfel de paşi - de citire sau de scriere - pot fi oriunde, chiar şi în interiorul ciclurilor.

Structura algoritmului este:



Să luăm ca exemplu 3 numere (**n ←--- 3**): **2, -5, 1** şi **a ←--- 0.** Deci problema este:

Cîte numere din cele trei sînt pozitive ?

pas 1: **a ←--- 0 , n ←--- 3**
pas 2: **(n ≤ 0) ≡ (3 ≤ 0) ≡ fals**
pas 3: **j ←--- 0**
pas 4: **i ←--- 1**
4.1: **x ←--- 2** (se citeşte în x primul număr din cele 3);
4.2: **(x>a) ≡ (2>0) ≡ adevărat ⇒ j ←--- j+1 ≡ 0+1 ≡ 1**
Ciclul s-a terminat, deci se revine la pasul 4:
pas 4: **i ←--- i+1 ≡ 1+1 ≡ 2**
(i≤n) ≡ (2≤3) ≡ adevărat şi deci se poate reintra în ciclu;
4.1: **x ←--- -5** (se citeşte următorul număr);
4.2: **(x>0) ≡ (-5>0) ≡ fals;** din nou
pas 4: **i ←--- i+1 ≡ 2+1 ≡ 3**
(i≤n) ≡ (3≤3) ≡ adevărat şi deci
4.1: **x ←--- 1** (se citeşte al treilea număr);
4.2: **(x>a) ≡ (1>0) ≡ adevărat ⇒ j ←--- j+1 ≡ 1+1 ≡2**
În sfîrşit,
pas 4: **i ←--- i+1 ≡ 3+1 ≡ 4**

(i≤n) ≡ (4≤3) ≡ fals (nu mai sînt alte numere de citit),
şi pasul 4 (bucla) este părăsit.

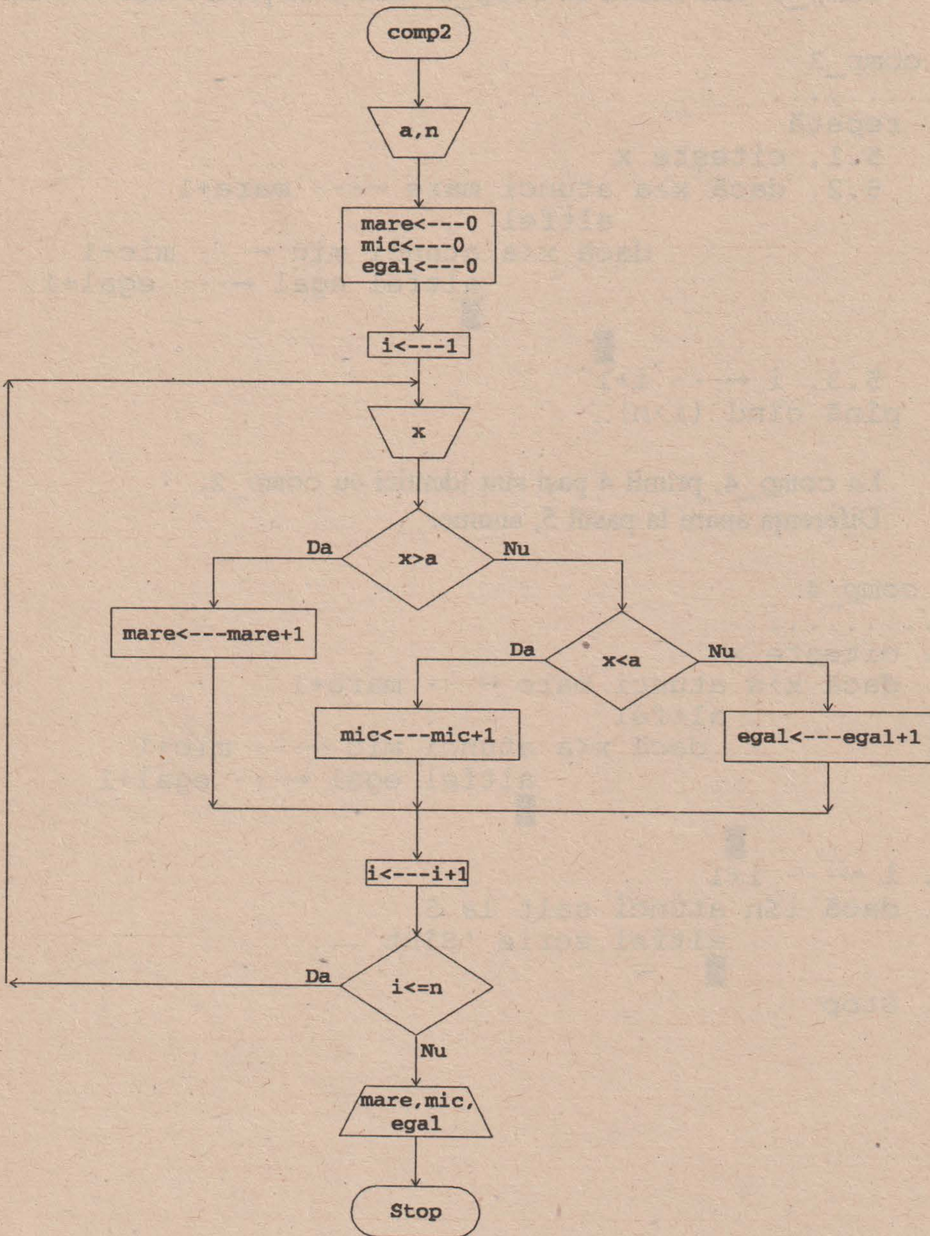
De remarcat că pasul 2 este doar o precauţie asupra numărului `n`; o valoare negativă pentru `n` ar face ca algoritmul să nu funcţioneze corect.

Să construim şi alte variante ale algoritmului, pe care îl vom completa rafinînd răspunsul. Astfel, în locul contorului `j` să introducem 3 variabile contor:

- `mare` pentru a număra cîte elemente sînt mai mari decît `a`;
- `mic`, pentru a număra cîte elemente sînt mai mici decît `a`;
- `egal`, pentru a da numărul elementelor egale cu `a`.

```
comp_2
a,x reale
n,i,mare,mic,egal  întregi
1. citeşte a,n
2. dacă n≤0 atunci scrie 'Eroare de date';Stop
                ▮
3. mare ←--- 0 ; mic ←--- 0 ; egal ←--- 0
4. i ←--- 1
5. cît timp i ≤ n
           execută
            5.1. citeşte x
            5.2. dacă x>a atunci mare ←--- mare+1
                          altfel
                      dacă x<a atunci mic ←--- mic+1
                               altfel egal ←---egal+1
                               ▮
                          ▮
            5.3. i ←--- i+1
           ▮
6. scrie 'Sînt ',mare,'componente mai mari decît ',a
         'Sînt ',mic,' componente mai mici decît ',a
         'Sînt ',egal,'componente egale cu ',a
7. Stop.
```

Variantele prezentate vor folosi în loc de operaţia `pentru` alte operaţii posibile. Structura de bază este însă aceeaşi, anume schema logică următoare:


comp2
a,n
mare<---0
mic<---0
egal<---0
i<---1
x
x>a
Da
Nu
mare<---mare+1
x<a
Da
Nu
mic<---mic+1
egal<---egal+1
i<---i+1
i<=n
Da
Nu
mare,mic,
egal
Stop

`comp_3` este identic cu `comp_2` înafară de pasul 5 care se scrie:

```
  comp_3
..........
5. repetă
    5.1. citeşte x
    5.2. dacă x>a atunci mare ←--- mare+1
                  altfel
                 dacă x<a atunci mic ←--- mic+1
                          altfel egal ←--- egal+1
                          ■
                  ■
    5.3. i ←--- i+1
   pînă cînd (i>n)
```

La `comp_4`, primii 4 paşi sînt identici cu `comp_2`.
Diferenţa apare la pasul 5, anume:

```
  comp_4
..........
5. citeşte x
6. dacă x>a atunci mare ←--- mare+1
            altfel
             dacă x<a atunci mic ←--- mic+1
                      altfel egal ←--- egal+1
                      ■
            ■
7. i ←--- i+1
8. dacă i≤n atunci salt la 5
            altfel scrie 'Sînt ...
            ■
9. Stop
```

3.1.3 Aflarea maximului (minimului) dintre n numere

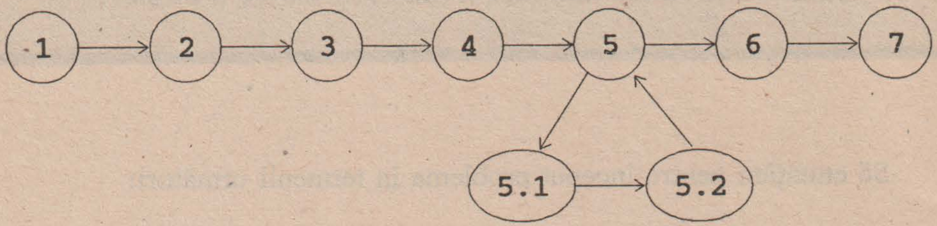
Să enunţăm pentru început problema în termenii următori:

*Se dau **n** numere reale. Să se afle cel mai mare dintre ele.*

Ca şi paragraful precedent, vom folosi două variabile de lucru: una (notată aici max) pentru a reţine cel mai mare număr, şi o alta - x, pentru a citi pe rînd cîte un număr din cele n.

```
maxim_1
max,x reale
n,i  întregi
1. citeşte n    /* numărul de elemente */
2. dacă n≤0 atunci scrie 'Eroare de date ';Stop
            █
3. citeşte max /* primul element este citit în max */
4. dacă n=1 atunci salt la 6
            █
5. pentru i ←--- 1,n-1
        5.1. citeşte x
        5.2. dacă x>max atunci max ←--- x
                         █
        █
6. scrie 'Cel mai mare număr este ',max
7. Stop
```

Structura algoritmului este :



Lăsăm ca exerciţiu construcţia schemei logice corespunzătoare.

De exemplu, fie **n ←--- 4** şi numerele **2, -3, 5, 0**. Atunci:

pas 1: **n ←--- 4**
pas 2: **(n ≤ 0) ≡ (4 ≤ 0) ≡ fals**
pas 3: **max ←--- 2** (primul număr este introdus în max)
pas 4: **(n = 1) ≡ (4 = 1) ≡ fals**
pas 5: **i ←--- 1**
  5.1: **x ←--- -3** (se introduce al doilea număr în x)
  5.2: **(x>max) ≡ (-3 > 2) ≡ fals**

Ciclul s-a încheiat, în max fiind **2**, cel mai mare dintre numerele **2, -3.**

pas 5: **i ←--- i+1 ≡ 1+1 ≡ 2**
  (i ≤ n-1) ≡ (2 ≤ 3) ≡ adevărat
  5.1: **x ←--- 5**
  5.2: **(x > max) ≡ (5 > 2) ≡ adevărat ⇒ max ←--- 5**

După această trecere prin ciclu, valoarea lui max s-a schimbat, fiind reţinut **5**, cel mai mare dintre numerele **2, -3, 5.**

pas 5: **i ←--- i+1 ≡ 2+1 ≡ 3**
  (i ≤ n-1) ≡ (3 ≤ 3) ≡ adevărat

5.1: **x ←--- 0** (se introduce ultimul element).

5.2: **(x > max) ≡ (0 > 5) ≡ fals**

Şi după această trecere prin buclă, max are valoarea **5**.

pas 5: **i ←--- i+1 ≡3+1 ≡ 4**

(i ≤ n-1) ≡ (4 ≤ 3) ≡ fals şi se iese din ciclu.

pas 6: **scrie 'Cel mai mare număr este 5'.**

Ciclul 5 se execută de `n-1` ori.

De remarcat că din cele `n` numere, primul se citeşte direct în `max` iar celelalte - pe rînd în `x`. Motivul este acela că la început este nevoie de o iniţializare a variabilei `max` (şi în mod evident, cel mai mare număr după citirea unuia singur, este acel număr). După ce se face această iniţializare, celelalte numere se citesc pe rînd în `x`; la fiecare parcurgere a ciclului de la pasul 5, acest element din `x` este comparat cu `max`, în care se află cel mai mare număr dintre toate cele citite anterior. Dacă `x` este mai mare decît `max`, înseamnă că el este mai mare decît toate numerele citite - şi deci trebuie reţinut în `max`.

Construcţia nu funcţionează în cazul limită **n ←--- 1**. Atunci, după citirea în `max`, algoritmul trebuie să treacă direct la scriere, deoarece nu mai sînt alte elemente de citit, numărul - unic - fiind evident şi maxim.

Această observaţie impune introducerea pasului 4 , care să elimine intrarea în ciclu pentru cazul **n ←--- 1**.

Totuşi, pasul 4 din `maxim1` poate fi evitat astfel:

```
3. citeşte max ; i ←--- 1
4. cît timp i≤n-1
        execută
        4.1. citeşte x
        4.2. dacă x>max atunci max ←--- x
                              ▮
        4.3. i ←--- i+1
        ▮
5. scrie 'Cel mai mare număr este ',max
6. Stop
```

Acum, dacă **n** ←--- **1**, atunci intrarea în ciclul 4 se face cu iniţializarea

```
i ←--- 1
```
şi testul
```
(i≤n-1) ≡ (1≤0) ≡ fals,
```
deci se trece direct mai departe la pasul 5.

Aparent, în această situaţie devine inutil şi testul de la pasul 2. Afirmaţia este falsă, pentru că dacă **n** ←--- **0**, atunci pasul 4 nu se execută; DAR, la pasul 3, ce se citeşte în `max` ? O variabilă care nu aparţine algoritmului şi care în final este declarată drept element maxim.

Putem construi algoritmul într-o formă mai omogenă, folosind ca iniţializare pentru `max` o valoare care nu poate fi niciodată maximă; o notăm -∞ şi ea este cea mai mică valoare numerică pe care o poate accepta calculatorul în realizarea algoritmului. Atunci:

```
maxim_2
max,x reale
n,i  întregi
1. citeşte n
2. dacă n≤0 atunci scrie 'Eroare de date';Stop
            ▮
3. max ←--- -∞
4. pentru i ←--- 1,n
        4.1. citeşte x
        4.2. dacă x>max atunci max ←--- x
                                ▮
        ▮
5. scrie 'Cel mai mare număr este',max
6. Stop
```

Acum precauţia de la pasul 4 din `maxim_1` nu mai este necesară, toate elementele (chiar dacă **n** ←--- **1**) citindu-se în cadrul ciclului.

Să construim un algoritm mai complex care să dea pentru `n` numere , elementul maxim, cel minim precum şi de cîte ori apar ele.

Vom folosi pentru aceasta variabilele:

`max` - pentru elementul maxim
`min` - pentru elementul minim
`kmax` - cîte elemente maxime sînt printre cele `n` numere
`kmin` - cîte elemente minime sînt printre cele `n` numere.

Celelalte variabile folosite au aceeaşi semnificaţie ca şi la algoritmul anterior.

```
maxmin_1
x,max,min  reale
n,i,kmax,kmin  întregi
1. citeşte n
2. dacă n≤0 atunci scrie 'Eroare de date';Stop
            ▮
3. max ←--- -∞ ; min ←--- ∞
4. kmax ←--- 0 ; kmin ←--- 0
5. pentru i ←--- 1,n
        5.1. citeşte x
        5.2. dacă max<x atunci
                         max ←--- x ; kmax ←--- 1
                        altfel
                  dacă max=x atunci kmax ←--- kmax+1
                             ▮
                        ▮
        5.3. dacă min>x atunci
                         min ←--- x ; kmin ←--- 1
                        altfel
                   dacă min=x atunci kmin ←--- kmin+1
                             ▮
                        ▮
        ▮
 6. scrie'Cel mai mare număr este ',max
        '.El apare de',kmax,'ori.'
        'Cel mai mic număr este ',min
        '.El apare de ',kmin,'ori.'
 7. Stop
```

Propunem o parcurgere a algoritmului pentru numerele **2, -3, 0, 2, 1** şi **n ←---5.**

Ca rezultat se va obţine

```
Cel mai mare număr este 2. El apare de 2 ori.
Cel mai mic număr este -3. Ele apare de 1 ori.
```

O variantă a sa folosind subalgoritmi este:

```
maxmin_2
x,max,min  reale
n,i,kmax,kmin  întregi
1. citeşte n
2. dacă n≤0 atunci scrie 'Eroare de date';Stop

3. max ←--- -∞ ; min ←--- ∞
4. kmax ←--- 0 ; kmin ←--- 0
5. pentru i ←--- 1,n
        5.1. citeşte x
        5.2. control(x,max,kmax,1)
        5.3. control(x,min,kmin,-1)

6. scrie 'Cel mai mare număr este ',max
        '.El apare de ',kmax,' ori.'
        'Cel mai mic număr este ',min
        '.El apare de ',kmin,' ori.'
7. Stop
  control(a,b,k,p)
  a,b  reale
  k,p  întreg
 1. dacă p*a>p*b atunci b ←--- a ; k ←--- 1
                   altfel
                 dacă a=b atunci k ←--- k+1


 2. Revenire
```

Construcţia s-a bazat pe observaţia că testul `min<x` se poate scrie `-min>-x` deci `-min` operează ca un maxim pentru elementele de forma `-x`. În acest fel, maximul şi minimul se pot determina cu un singur subalgoritm.

Să verificăm `maxmin_2` pentru numerele **2, -3, 2.**

pas 1: **n ←--- 3**
pas 2: **(n≤0) ≡ (3≤0) ≡ fals**
pas 3: **max ←--- -∞; min ←--- ∞**
pas 4: **kmax ←--- 0 ; kmin ←--- 0**
pas 5: **i ←--- 1**
5.1: **x ←--- 2**
5.2: `control(x,max,kmax,1)`
Algoritmul `control` va lucra cu variabilele
a ←--- x ≡ 2, b ←--- max ≡ -∞, k ←--- kmax ≡ 0, p←---1.
P.1: **(p*a>p*b) ≡ (2>-∞) ≡ adevărat**
b ←--- 2, k ←--- 1
Deci revenirea în algoritmul principal se face cu
max ←--- 2, kmax ←--- 1.
5.3: `control(x,min,kmin,-1)`
Operarea cu **a ←--- x ≡ 2, b ←--- min ≡ ∞, k ←--- kmin ≡ 0, p←--- -1** în subalgoritm conduce la:
P.1: **(p*a>p*b) ≡ (-2>-∞) ≡ adevărat**
b ←--- 2, k ←--- 1
Deci **min ←--- 2** şi **kmin ←--- 1.**
pas 5: **i ←--- i+1 ≡ 1+1 ≡ 2**
(i≤n) ≡ (2≤3) ≡ adevărat
5.1: **x ←--- -3**
5.2: `control(x,max,kmax,1)` a cărui efectuare nu duce la nici o modificare a datelor.
5.3. `control(x,min,kmin,-1)`

Se intră în subalgoritm cu **a←--- -3, b←--- 2, k←--- 1, p←--- -1**. Atunci

P.1: **(p*a>p*b) ≡ (3>-2) ≡ adevărat**

b ←--- a ≡ 3, k ←--- 1

`Revenire` are ca efect

min ←--- -3, kmin ←--- 1.

pas 5: **i ←--- i+1 ≡ 2+1 ≡ 3**

(i≤n) ≡ (3≤3) ≡ adevărat

5.1: **x ←--- 2**

5.2: `control(x,max,kmax)`

Deci subalgoritmul debutează cu iniţializările

a ←--- 2, b ←--- max ≡ 2, k ←--- kmax ≡ 1.

P.1: **(a>b) ≡ (2>2) ≡ fals**

(a=b) ≡ (2=2) ≡ adevărat ⇒ **k ←--- k+1 ≡ 2**

P.2: `Revenire:` **max ←--- b ≡ 2, kmax ←--- k ≡ 2**

5.3. `control(x,min,kmin,-1)` care nu modifică nimic.

pas 5: **i ←--- i+1 ≡ 3+1 ≡ 4**

(i≤n) ≡ (4≤3) ≡ fals şi bucla 5 este abandanonată.

pas 5: **scrie: Cel mai mare număr este 2. El apare de 2 ori.**

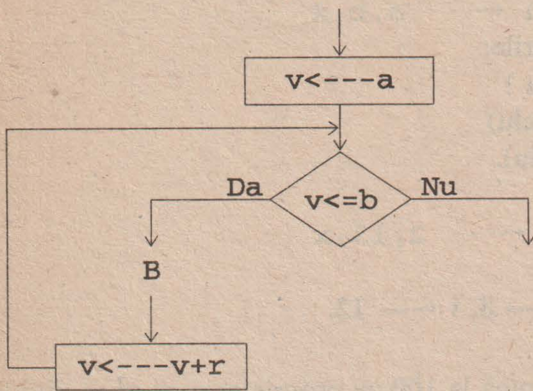
Cel mai mic număr este -3. El apare de 1 ori.

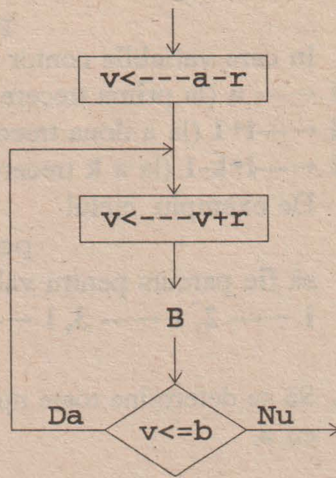
EXERCIŢII

1. Să se scrie detaliat algoritmii `comp_3` şi `comp_4` şi să se verifice cu diverse date de intrare care să treacă prin toate variantele posibile.

2. Să se justifice valabilitatea expresiei (*) .

3. Cum se traduc în paşi de algoritm buclele





4. Să se determine produsul numerelor naturale de la **1** la **n**.

5. Se dă un polinom **p(x)** şi o valoare reală **a**. Să se calculeze **p(a)**.

6. Să se treacă un număr din baza **b** în baza 10.

7. Cîţi divizori pozitivi are un număr **n** ? Să se decidă apoi dacă **n** este prim.

8. Să se dea exemplu de construcţie în care ciclul

```
pentru  i ←--- a,b
```

nu se termină.

9. Ce condiţii verifică `a` şi `b` pentru ca ciclul

```
pentru i ←--- a,b,-1
      x ←--- x+a
            ■
```

să nu se termine.

10. Să se imagineze un ciclu

```
pentru i ←--- a,b,x
```

în care variabila contor să ia valorile:

i ←--- a (la prima trecere prin ciclu)
i ←---i+1 (la a doua trecere prin ciclu)
i ←---i+k-1 (la a **k** trecere prin ciclu).

De exemplu, ciclul

```
pentru i ←--- 2,13,x
```

să fie parcurs pentru valorile
i ←--- 2, i ←--- 3, i ←--- 5, i ←--- 8, i ←--- 12.

11. Să se determine toate numerele **a** pînă la 100 cu proprietatea că $\mathbf{a}^2$ se divide cu **a**.

12. Să se determine toate tripletele de numere **a,b,c** cu proprietățile:
 i. **1 < a < b < c < 100**
 ii. **a+b+c** se divide cu **10.**

13. Să se determine toate tripletele de numere **a,b,c** cu proprietățile:
 i. **1 < a < b < c < 100**
 ii. $\mathbf{a^2+b^2=c^2}$ (tripletele pitagoreice).

14. Se dau două numere întregi **a** și **b.** Să se scrie în ordine crescătoare toate numerele care se divid cu **a** sau cu **b** și sînt mai mici decît un număr dat **c.**

15. Se dă secvența de numere: 1 3 9 27 81 ... și un număr întreg **k.** Să se scrie al **k**-lea număr care urmează în secvență.

16. Aceeași problemă pentru secvențele
 1 1 2 3 5 8 13 21 ...
 2 6 10 14 18 ...

17. Se dă o zi a săptămînii. Să se listeze toţi anii din secolul XX care încep în acea zi a săptămînii. Se presupune cunoscută ziua în care cade 1 Ianuarie 1993.

18. Se dă o dată (**zi, lună, an**) din secolul XX. Să se calculeze cîte zile au trecut de la 1 Ianuarie 1900 pînă la această dată. Dar cîte săptămîni ?

3.2 VARIABILE DE TIP TABLOU

Să ne gîndim la următoarea situaţie:

În triajul unei gări se află vagoane; vagoane de călători, poştale, restaurant, de dormit, etc. Din toată această mulţime amorfă de vagoane, se selectează o parte cu care se compune un tren.

Prin ce se deosebesc cele două momente ?

- În prima fază, fiecare vagon este independent, fără nici o legătură cu celelalte obiecte aflate în vecinătate. Poate fi manevrat, încărcat şi aşezat oriunde în incinta triajului.
- După formarea garniturii de tren, vagoanele cuprinse în ea nu mai sînt independente; fiecare are unul sau doi vecini cu care împarte aceleaşi proprietăţi (poartă acelaşi nume - numărul trenului, merg în aceeaşi direcţie, se opresc în acelaşi loc, etc). Din acel moment proprietăţile vagoanelor, deşi se păstrează (sînt tot vagoane de clasă, poştale, de dormit, restaurant), se supun unei comenzi superioare, compunînd o nouă entitate - aceea de tren.

Aceeaşi situaţie o avem şi în cazul variabilelor.

În general sînt variabile independente - reale, întregi, caracter, etc. - care se comportă liber unele faţă de altele.

Sînt însă momente cînd este avantajos să legăm anumite grupuri de variabile ca să formeze entităţi noi, cu proprietăţi comune.

ORICE GRUP DE VARIABILE CU PROPRIETĂŢI COMUNE ŞI CU UN SINGUR NUME SE NUMEŞTE TABLOU.

Un tablou se caracterizează prin:

i. tipul variabilelor sale

TOATE VARIABILELE UNUI TABLOU AU ACELAŞI TIP.

Acesta poate fi întreg, real, caracter, logic, sau tot un tablou.

ii. dimensiunea

Un tablou poate avea una sau mai multe dimensiuni. Dacă elementele unui tablou sînt variabile independente, tabloul are o dimensiune. Aceasta se declară la începutul algoritmului (înainte de utilizarea lui) printr-o informaţie de forma

`tablou <nume>[k] de <tip>`

unde:

<nume> este numele general dat tabloului

`k` - numărul de variabile cuprinse în tablou

<tip> - tipul variabilelor

De exemplu,

`tablou a[3] de întregi`

desemnează trei variabile întregi avînd acelaşi nume - `a`. Dintre ele, `a[1]` este prima variabilă, `a[2]` - a doua şi `a[3]` - ultima.

În interiorul algoritmului, `a[i]` se foloseşte ca variabilă independentă (cu toate proprietăţile corespunzătoare).

Cînd componentele unui tablou de dimensiune **1** sînt tablouri de dimensiune **k**, atunci tabloul este de dimensiune **k+1**.

Pentru a facilita înţelegerea, vom lucra numai cu tablouri de dimensiune **k** ale căror componente sînt **tablouri cu structură identică**. Se poate considera, ca un caz limită, că orice variabilă independentă este un tablou de dimensiune 0.

De exemplu, fie tabloul `tab` cu trei componente tablouri de dimensiune 1, avînd toate cîte 4 elemente reale. Numele acestor componente nu mai interesează fiind caracterizate de numele general `tab`, care va fi definit

```
tablou tab[3,4] de reale
```

Prin aceasta înţelegem că `tab` este un tablou de dimensiune 2 cu 3 componente, fiecare componentă fiind un tablou cu 4 elemente numere reale.

Aceste elemente, care se folosesc în algoritm, sînt:

```
tab[1,1]    tab[1,2]    tab[1,3]    tab[1,4]
tab[2,1]    tab[2,2]    tab[2,3]    tab[2,4]
tab[3,1]    tab[3,2]    tab[3,3]    tab[3,4]
```

Fiecare `tab[i,j]` se comportă în algoritm ca o variabilă reală independentă.

Atenţie Să nu se facă confuzie între `tab[3,4]` care apare în zona de declarări şi eventuala apariţie a caracterului `tab[3,4]` în interiorul algoritmului. În definiţie, `tab[3,4]` arată că există 3*4=12 variabile reale avînd toate acelaşi nume `tab`, aşezate într-un tablou de dimensiune 2.

În algoritm, `tab[3,4]` se referă la una din aceste 12 variabile, anume la cea de coordonate **(3,4)**.

Observaţie: După cum am văzut, orice variabilă x poate fi definită ca un tablou cu un element `x[1]`. Pentru a evita confuziile (inerente aici), vom pune condiţia ca într-un algoritm **să nu existe simultan variabile simple şi tablouri cu acelaşi nume**.

Să prezentăm ca exemple cîţiva algoritmi elementari care folosesc tablouri.

3.2.1 Anularea elementelor unui tablou

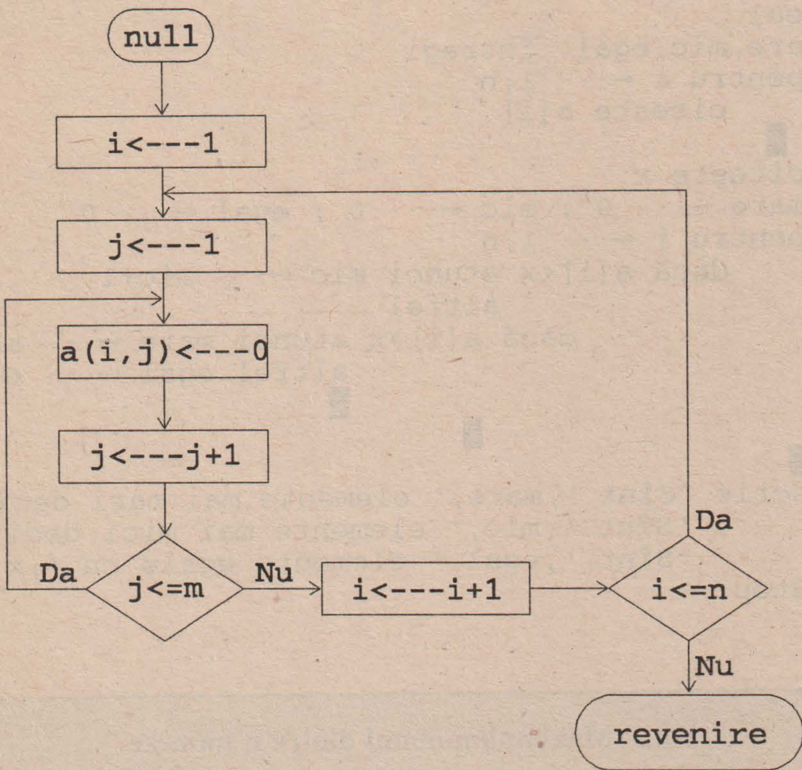
Problema este foarte des folosită sub forma unui subalgoritm care anulează primele n elemente ale unui tablou de dimensiune 1 cu componente întregi sau reale.

```
null(n)
tablou a[n] de întregi  /* sau reale  */
i întreg
1. pentru i ←--- 1,n
        a[i] ←--- 0
   ▌
2. Revenire
```

În cazul unui tablou de dimensiune 2, problema este la fel de simplu de rezolvat:

```
null(n,m)
tablou a[n,m] de întregi /* sau reale */
i,j  întregi
1. pentru i ←--- 1,n
        pentru j ←--- 1,m
           a[i,j] ←--- 0
        ▌
   ▌
2. Revenire
```

Schema logică arată ceva mai complicat:



3.2.2 Cîte elemente dintr-un şir sînt mai mari/mici decît x

Anterior am rezolvat această problemă fără a folosi tablouri, prin introducerea treptată a elementelor şirului; în acest fel însă, elementele se pierd şi nu mai pot fi găsite pentru o altă eventuală utilizare. Vom da o soluţie folosind datele introduse de la început într-un tablou.

```
comp_5
tablou a[n] de reale
x real
i,mare,mic,egal  întregi
1. pentru i ←--- 1,n
       citeşte a[i]
   █
2. citeşte x
3. mare ←--- 0 ; mic ←--- 0 ; egal ←--- 0
4. pentru i ←--- 1,n
       dacă a[i]<x atunci mic ←--- mic+1
                   altfel
            dacă a[i]>x atunci mare ←--- mare+1
                        altfel egal ←--- egal+1
                        █
                █
   █
5. scrie 'Sînt ',mare,' elemente mai mari decît ',x
         'Sînt ',mic,' elemente mai mici decît ',x
         'Sînt ',egal,' elemente egale cu ',x
6. Stop
```

3.2.3 Maximul/minimul dintre n numere

Şi această problemă poate fi tratată în mod natural începînd cu introducerea tuturor datelor într-un tablou pentru a le putea folosi ulterior.

Facem menţiunea că o astfel de utilizare duce la o risipă de memorie, inutilă dacă în problemă nu mai sînt necesare datele de intrare.

```
maxmin_3(n)
tablou x[n] de reale
max,min  reale
n,i,kmax,kmin  întregi
```

```
1. max ←--- -∞ ; min ←--- ∞
2. kmax ←--- 0 ; kmin ←--- 0
3. pentru i ←--- 1,n
       citeşte x[i]
   █
4. pentru i ←--- 1,n
       4.1. control( x[i],max,kmax,1)
       4.2. control(x[i],min,kmin,-1)
   █
5. scrie 'Cel mai mare număr este ',max
         '.El apare de ',kmax,' ori'
         'Cel mai mic număr este ',min
         '.El apare de ',kmin,' ori'
6. Stop
```

`control(a,b,k,p)` ... este subalgoritmul definit în algoritmul `maxmin_2` dat anterior (pag. 110).

Avantajul folosirii tabloului de date `x` este acela că putem determina şi poziţia pe care se află elementele extreme. Să construim un astfel de algoritm.

```
maxmin_4(n)
tablou a[n] de reale
tablou maxim[n], minim[n] de întregi
max, min  reale
i, kmax, kmin  întregi
1. pentru i ←--- 1,n
       citeşte a[i]
   █
2. kmax ←--- 1; kmin ←--- 1;
   maxim[1] ←--- 1; minim[1] ←--- 1
3. max ←--- a[1]; min ←--- a[1]; i ←--- 2
4. cît timp i≤ n
        execută
   4.1. dacă a[i]>max atunci
            kmax ←--- 1; max ←--- a[i]; maxim[1] ←--- i
                      altfel
                 dacă a[i]=max atunci kmax ←--- kmax+1;
                                      maxim[kmax] ←--- i
```

```
                                    ▮
                        ▮
   4.2. dacă a[i]<min atunci
            kmin ←--- 1; min ←--- a[i]; minim[1] ←--- i
                        altfel

                   dacă a[i]=min atunci kmin ←--- kmin+1;
                                      minim[kmin] ←--- i
                                   ▮
                        ▮
   4.3. i ←--- i+1
   ▮
 5. scrie ' Elementul maxim este ',max
          '.El se află în tablou pe poziţiile :'
   5.1. pentru i ←--- 1,kmax
             scrie maxim[i]
          ▮
 6. scrie ' Elementul minim este ',min
          '.El se află în tablou pe poziţiile :'
   6.1. pentru i ←--- 1,kmin
             scrie minim[i]
          ▮
 7. Stop
```

Elementele care trebuiesc analizate sînt în tabloul `a`. `kmax` şi `kmin` dau numărul de elemente maxime respectiv minime. În tabloul `maxim` se introduc poziţiile pe care se află cele `kmax` elemente maxime egale, iar în tabloul `minim` se introduc cele `kmin` poziţii ale elementelor minime egale.

Să exemplificăm algoritmul pentru tabloul **a=(1.1, 12, -3, 12, 0)**.
Aici **n ←---5**.

pas 1: **a[1] ←--- 1.1, a[2] ←--- 12, a[3] ←--- -3, a[4] ←--- 12, a[5] ←--- 0**
pas 2: **kmax ←--- 1, kmin ←--- 1, maxim[1] ←--- 1, minim[1] ←--- 1**
pas 3: **max ←--- a[1] ≡ 1.1, min ←--- a[1] ≡ 1.1, i ←--- 2**
pas 4: **(i≤n) ≡ (2≤5) ≡ adevărat**
   4.1: **(a[i]>max) ≡ (12>1.1) ≡ adevărat**

kmax ←--- 1, maxim[1] ←--- 2, max ←--- 12

Pentru primele două componente există un maxim **12** aflat pe poziţia a doua.

4.2: **(a[i]<min) ≡ (12<1.1) ≡ fals**

(a[i]=min) ≡ (12=1.1) ≡ fals

Pentru primele două componente ale tabloului, minimul este **1**, aflat pe prima poziţie.

4.3: **i ←--- i+1 ≡ 2+1 ≡ 3**

pas 4: **(i≤n) ≡ (3≤5) ≡ adevărat**

4.1: **(a[i]>max) ≡ (-3>12) ≡ fals**

(a[i]=max) ≡ (-3=12) ≡ fals

4.2: **(a[i]<min) ≡ (-3<1.1) ≡ adevărat**

min ←--- -3, kmin ←--- 1, minim[1] ←--- 3

Pentru primele 3 elemente ale tabloului există un singur minim **-3** aflat pe poziţia a treia.

4.3: **i ←--- i+1 ≡ 3+1 ≡ 4**

pas 4: **(i≤n) ≡ (4≤5) ≡ adevărat**

4.1: **(a[i]>max) ≡ (12>12) ≡ fals**

(a[i]=max) ≡ (12=12) ≡ adevărat

kmax ←--- kmax+1 ≡ 1+1 ≡ 2, maxim[2] ←--- 4

Printre primele 4 componente ale tabloului sînt două maxime situate pe poziţiile 2 şi 4.

4.2: **(a[i]<min) ≡ (12<-3) ≡ fals**

(a[i]=min) ≡ (12=-3) ≡ fals

4.3: **i ←--- i+1 ≡ 4+1 ≡ 5**

pas 4: **(i≤n) ≡ (5≤5) ≡ adevărat**

4.1: **(a[i]>max) ≡ (0>12) ≡ fals**

(a[i]=max) ≡ (0=12) ≡ fals

4.2: **(a[i]<min) ≡ (0<-3) ≡ fals**

(a[i]=min) ≡ (0=-3) ≡ fals

4.3: **i ←--- i+1 ≡ 5+1 ≡ 6**

pas 4: **(i≤n) ≡ (6≤5) ≡ fals**

pas 5: **Elementul maxim este 12.**

El se află în tablou pe poziţiile 2 4

pas 6: **Elementul minim este -3.**
El se află în tablou pe poziţiile 3

Evident, algoritmul poate fi îmbunătăţit.
De exemplu, o facilitare ar putea fi făcută adăugînd la pasul 4.1., după `maxim[1] ←--- i`, operaţia

```
salt la 4.3.
```

Lăsăm ca exerciţiu justificarea acestei optimizări.

EXERCIŢII

1. Să se rescrie algoritmii `maxim_1`, `maxim_2` şi `maxmin_1` folosind tablouri.

2. Se dă un număr **n** . Să se introducă cifrele sale într-un tablou.
De exemplu, pentru numărul **31** se va obţine
`nr[1] ←--- 3, nr[2] ←--- 1.`

3. Se dă un tablou cu **n** numere. Să se determine:
i. Cîte numere sînt pare;
ii. Cîte numere sînt intr-un interval dat [a,b].

4. Se dau **n** numere (n≥3). Să se afle cele mai mari 3 numere dintre acestea.

5. Să se calculeze 2^n pentru **n** între 31 şi 50. Se ştie că pentru aceste valori, implementarea algoritmului obişnuit (prin înmulţiri) pe un calculator conduce la numere prea mari - deci eroare. Să se construiască un algoritm care să elimine acest neajuns.

6. Să se cerceteze dacă un nume este **palindrom**. (Un nume este **palindrom** dacă se citeşte la fel de la stînga la dreapta cît şi de la dreapta la stînga . Exemple: capac, sos, potop, epuraşul uşa rupe).

7. Să se determine numerele prime pînă la 100 folosind ciurul lui Eratostene.

8. Se dă un tablou **a[n,n]**. Să se facă zero
   - toate elementele aflate pe diagonala principală;
   - toate elementele situate sub diagonala principală;
   - toate elementele situate deasupra diagonalei principale.

   (Elementele **a[1,1]**, **a[2,2],...,a[n,n]** formează diagonala principală a tabloului).

9. Se dă un tablou **a[n]**. Să se construiască un tablou **b[n]** definit astfel: **b[1] ←--- a[1], b[2] ←--- a[1]+a[2],...,b[n] ←--- a[1]+a[2]+..a[n].**

10. Se dă un tablou **a[n]**. Să se elimine toate elementele care se repetă.

11. Să se construiască algoritmi care să realizeze:
    - Reuniunea a două mulţimi;
    - Intersecţia a două mulţimi;
    - Diferenţa a două mulţimi.

12. Se dă un tablou care conţine o propoziţie. Să se determine cîte cuvinte sînt în acea propoziţie.

13. În aceeaşi ipoteză de mai sus, să se determine cîte cuvinte din propoziţie încep cu o literă dată. Dar cîte cuvinte sînt formate din două litere ?

3.3 ALGORITMI CICLICI CU NUMĂR NECUNOSCUT DE PAŞI

In această categorie intră algoritmii în care buclele sînt parcurse de un număr neprecizat de ori. Variabilele de tip contor - dacă există - nu folosesc la numărarea trecerilor prin ciclu ci doar pentru a facilita verificarea unor condiţii.

Dintre operaţiile iterative definite anterior, cea care nu poate fi folosită este `pentru...`

În această clasă apar în general cele mai dificile tipuri de algoritmi, care pot combina în cadrul lor toate genurile de algoritmi prezentaţi.

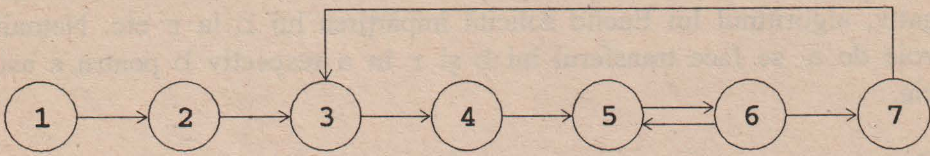
3.3.1 Algoritmul lui Euclid

Este cel mai cunoscut algoritm folosit în matematică (a fost o perioadă cînd termenul de algoritm desemna numai algoritmul lui Euclid).

Problema se enunţă astfel:

*Pentru două numere naturale **a** şi **b** (a≥b) să se calculeze cel mai mare divizor comun (cmmdc) dintre **a** şi **b**.*

```
euclid_1
a,b,r  întregi
1. citeşte a,b
2. dacă a*b=0 atunci scrie 'Eroare de date'; Stop
3. dacă a<b atunci r ←--- a ; a←--- b ; b ←--- r
4. dacă a=b atunci salt la 7
5. r ←--- a-a/b*b
6. dacă r≠0 atunci a ←--- b ; b ←--- r ; salt la 5
7. scrie b ; Stop
```



Vom face următoarele comentarii asupra acestui algoritm `euclid_1`:

ⓐ Ciclul pas 5 ⇔ pas 6 se execută fără nici un contor, doar pe baza expresiei relaţionale $r \neq 0$ (echivalentă aici cu $r > 0$). Numărul de parcurgeri ale ciclului nu este cunoscut; matematic însă se demonstrează că el este finit.

ⓑ pasul 2: deoarece la pasul 5 se face o împărţire, testăm dacă vreunul din numerele a sau b este zero. În caz afirmativ, înseamnă că datele au fost introduse greşit. Aici algoritmul se poate opri (`Stop`) sau se poate solicita un alt set de date (`salt la 1`) obţinîndu-se în diagramă o altă buclă cu număr neprecizat de paşi:

pas 1 ⇔ pas 2

Pasul 3 elimină posibilitatea $a<b$ (cînd ar trebui să calculăm restul împărţirii lui b la a), permutîndu-le (cu "regula celor trei pahare ") şi ţinînd cont că

cmmdc(a, b) = **cmmdc**(b, a)

La pasul 4 se elimină cazul particular cînd $a=b$, trecînd direct la scriere. El poate lipsi din algoritm, pasul 5 dînd din prima trecere valoarea **adevărat** la testul $a=b$. De asemenea, dacă la pasul 3 s-a permutat a cu b, se poate sări peste pasul 4 (testul $a=b$ va avea valoarea **fals**) adăugînd după b ←-- - r operaţia

```
salt la 5
```

Pasul 5 calculează restul împărţirii lui a la b folosind un subalgoritm liniar (`rest`- pag. 59).

Pasul 6 testează dacă împărţirea lui `a` la `b` s-a făcut exact. În caz negativ, algoritmul lui Euclid solicită împărţirea lui `b` la `r` etc. Nemaifiind nevoie de `a`, se face transferul lui `b` şi `r` în `a` respectiv `b` pentru a asigura bucla.

ⓒ Dacă pasul 3 lipseşte din algoritm, acesta va continua să dea rezultate corecte !

Într-adevăr, dacă la pasul 1 se citesc `a` şi `b` cu `a<b` şi se ignoră pasul 3, se ajunge la pasul 5 unde obţinem `r ←--- a` (deoarece `a/b=0`).
La pasul 6 se fac asignările `a ←--- b` şi `b ←--- r`, deci se completează regula celor trei pahare obţinîndu-se acelaşi efect ca la pasul 3.

ⓓ Variabila auxiliară utilizată în `regula celor trei pahare` de la pasul 3 este aceeaşi în pasul 5 (unde se calculează restul). Cei doi paşi fiind distincţi, nu apare nici o confuzie de notaţie.

ⓔ Cum pasul 3 se poate elimina, se pune întrebarea dacă nu cumva variabila de lucru `r` este inutilă (şi ne putem dispensa de ea). Răspunsul este afirmativ, dar ceva mai dificil de construit.

Într-adevăr, se pare că la prima vedere, paşii 5 şi 6 se pot restrînge în

```
5. cît timp a≠a/b*b
            execută
                 a ←--- b
                 b ←--- a-a/b*b
            █
```

Afirmaţia este falsă, ceea ce se poate vedea imediat pe un exemplu (de fapt nu se verifică `regula celor trei pahare`).

Algoritmul corect va fi

```
euclid_2
a,b  întregi
1. citeşte a,b
```

```
2. dacă a*b=0 atunci scrie 'Eroare de date'; Stop
           █
3. cît timp b≠0
        execută
         a ←--- a+b-a/b*b; b ←--- a-b; a ←--- a-b
        █
4. scrie a
5. Stop
```

Într-adevăr, să vedem ce rezultat va da `euclid_2` pentru **a ←--- 12, b ←--- 18.** Repetăm:

ACEASTA ESTE O VERIFICARE, NU O DEMONSTRAŢIE A CORECTITUDINII ALGORITMULUI EUCLID2.

pas 1: **a ←--- 12 , b ←--- 18**
pas 2: **(a*b=0) ≡ (12*18 = 0) ≡ fals**
pas 3: **(b≠0) ≡ (18≠0) ≡ adevărat**
 a ←--- a+b-a/b*b ≡ 12+18-12/18*18 ≡ 30
 b ←--- a-b ≡ 30-18 ≡ 12
 a ←--- a-b ≡ 30-12 ≡ 18
Deci, o permutare a lui `a` cu `b` fără nici o variabilă auxiliară.
Se revine la pasul 3 cu **a ←--- 18** şi **b ←--- 12.**
pas 3: **(b ≠ 0) ≡ (12 ≠ 0) ≡ adevărat**
 a ←--- a+b-a/b*b ≡ 12+18-18/12*12 ≡ 30-12 ≡ 18
 b ←--- a-b ≡ 18-12 ≡ 6
 a ←--- a-b ≡ 18-6 ≡ 12
Noile valori sînt **a ←--- 12** şi **b ←--- 6.**

pas 3: **(b≠0) ≡ (6≠0) ≡ adevărat**
 a ←--- a+b-a/b*b ≡ 12+6-12/6*6 ≡ 18-12 ≡ 6
 b ←--- a-b ≡ 6-6 ≡ 0
 a ←--- a-b ≡ 6-0 ≡ 6
pas 3: **(b≠0) ≡ (0≠0) ≡ fals**
şi se scrie `a` care are valoarea **6 = cmmdc(12,18).**

Să prezentăm şi o variantă mai simplă a algoritmului lui Euclid:

```
euclid_3
x,a,b  întregi
1. citeşte a,b
2. dacă a*b=0 atunci scrie 'Eroare de date';salt la 1
                    ■
3. cît timp a>b
            execută
                a ←--- a-b
            ■
4. dacă a=b atunci scrie b;Stop
            altfel
                x ←--- a;a ←--- b;b ←--- x;salt la 3
               ■
```

Este algoritmul lui Euclid scris cu scăderi în loc de împărţiri, ţinîndu-se cont de relaţia

cmmdc(a,b) = cmmdc(a-b,b)

Prin scrierea repetată a celor două numere (cu grija de a le permuta cînd primul devine mai mic), se ajunge la un moment cînd ele devin egale.

Atunci

cmmdc(a,a)=a

Cîteva observaţii:

a) Orice algoritm se termină cu `Stop` (sau `Revenire`). Totuşi, este posibil ca această operaţie să nu apară pe ultima poziţie la scriere. `Stop` *este ultima operaţie care se execută* !

Dacă dorim ca ea să apară pe ultima poziţie în `euclid_2`, modificăm pasul 4 astfel:

```
4. dacă a≠b atunci
                x ←--- a;a ←--- b;b ←--- x;salt la 3
             altfel scrie b
             ■
5. Stop
```

b Să presupunem că nu introducem testarea de control `b=0`. În variantele anterioare, `euclid_1` sau `euclid_2` se blochează la împărţire anunţînd

'Număr prea mare '.

În varianta `euclid_3`, pasul 3 se transformă în ciclu infinit, pentru că testul `a>0` este totdeauna **adevărat** şi se efectuează asignarea

`a ←--- a-0.`

c Paşii 3 şi 4 se pot scrie şi

```
3. dacă a>b atunci  a←--- a-b
          altfel
       dacă a<b atunci x ←--- a;a ←--- b;b ←--- x
                altfel scrie b;Stop
                █
          █
4. salt la 3
```

d Pentru o reluare a algoritmului cu alte perechi de numere, înlocuim `Stop` cu `salt la 5` şi adăugăm

```
5. scrie 'Reluăm algoritmul ?',s
6. dacă s='da' atunci salt la 1
               altfel Stop
               █
```

unde `s` este o variabilă formată dintr-un şir de caractere, care trebuie declarată de la început.

e Şi în acest caz, permutarea lui `a` cu `b` se poate elimina, construindu-se algoritmul

```
   euclid_4
   a,b  întregi
1. citeşte a,b
2. dacă a*b=0 atunci scrie 'Eroare de date';
                           salt la 1
              █
```

```
3. cît timp a≠b
          execută
               dacă a<b  atunci b ←--- b-a
                         altfel a ←--- a-b
                         █
          █
4. scrie a
5. Stop
```

În plus, euclid_4 este mult mai rapid decît ceilalţi algoritmi prezentaţi anterior.

3.3.2 Testarea dacă un număr este prim

Se ştie că un număr n este prim dacă nu are alţi divizori pozitivi înafară de **1** şi **n**. Orice divizor diferit de **+1, -1, +n,-n** este un divizor propriu.

```
prim_1
i,n  întregi
1. citeşte n
2. i ←--- 2
3. cît timp i≤n/2
          execută
               dacă n=n/i*i atunci salt la 5
                            altfel i ←--- i+1
                            █
          █
4. scrie n' este număr prim' ; Stop
5. scrie n' nu este număr prim' ; Stop
```

Algoritmul, foarte simplu, testează dacă n admite divizori în intervalul **[2,n/2]**.

De remarcat că deşi există un contor i, numărul de treceri prin ciclul format de pasul 3 nu este cunoscut deoarece în momentul cînd se găseşte un divizor, acest ciclu este părăsit. Putem determina numai numărul maxim posibil de utilizări ale ciclului: [n/2]+1.

Este interesant că acest algoritm poate fi transformat într-unul cu număr cunoscut de paşi. Pentru aceasta vom folosi o variabilă suplimentară numită **indicator** care în final va lua valoarea **0** dacă numărul n nu este prim, sau **1** - dacă n este prim.

Ciclul va fi parcurs de exact **max {[n/2]-1,1}** ori.

Dezavantajul acestui algoritm va fi acela că după ce găsim un divizor al lui n, deşi facem indicatorul să ia valoarea **0**, vom continua verificările în buclă, cu toate că ele nu mai sînt necesare.

```
prim_2
n,i,p  întregi
1. citeşte n
2. p ←--- 1
3. dacă n≤1 atunci scrie 'Eroare de date ';
                   salt la 1
            altfel dacă n=2 atunci salt la 5
                               ■
            ■
4. pentru i ←--- 2,n/2
        dacă n=n/i*i atunci p ←--- 0
                      ■
        ■
5. dacă p=0 atunci scrie n' nu este număr prim'
            altfel scrie n' este număr prim'
            ■
6. Stop
```

De exemplu pentru **n ←--- 9000**, deşi de la prima trecere (la împărţirea cu **2**) se decide că n nu este prim, ciclul continuă să fie executat de încă **4498** ori în mod inutil.

Dacă ne punem problema de eficienţă a algoritmului, o variantă care să decidă răspunsul mai rapid se poate construi plecînd de la următoarea proprietate (a cărei demonstraţie o propunem ca exerciţiu):

(*) Dacă n admite un divizor propriu pozitiv,
atunci admite un divizor propriu ≤ rădăcina(n).

```
prim_3
 n,i,p  întregi
 1. citeşte n
 2. dacă n≤1 atunci scrie 'Eroare de date';salt la 1
                ■
 3. dacă n≤3 atunci salt la 7
                ■
 4. dacă n=n/2*2 atunci salt la 8
                    ■
 5. p ←--- rădăcina(n) ; i ←--- 3

 6. cît timp i≤p
            execută
               dacă n=n/i*i atunci salt la 8
                            altfel i ←--- i+2
                            ■
            ■
 7. scrie n' este număr prim'; Stop
 8. scrie n' nu este număr prim'; Stop
```

Să facem cîteva comentarii asupra acestui algoritm:

a. Aproape jumătate din paşii lui o formeză testele de început; astfel

♦ pasul 2 asigură numere naturale nenule la intrare; în caz contrar se anunţă eroare şi se aşteaptă asignarea altei valori pentru n.
De remarcat că se formează o buclă pas 1 ⇔ pas 2
din care se iese numai dacă n primeşte o valoare strict mai mare ca **1**.

♦ condiţia n≤3 este în acest context echivalentă cu

(n=2) sau (n=3)

În caz afirmativ - cele două numere sînt prime - se trece direct la pasul 7. În particular se elimină astfel numărul **2**, singurul număr prim par.

♦ pasul 4 studiază dacă `n` este număr par. Avînd în vedere că jumătate din testele algoritmului anterior erau relative la împărţiri cu numere pare, dacă n nu este divizibil cu **2**, celelalte verificări sînt inutile.

♦ De fapt paşii 2-4 se pot scrie într-unul singur astfel:

```
2'.dacă n≤1 atunci scrie 'Eroare de date';salt la 1
            altfel
              dacă n≤3 atunci salt la 7
                         altfel
                       dacă n=n/2*2 atunci salt la 8
                                         ▮
                             ▮
            ▮
```

b. Variabila contor `i` nu are pasul **1** ci **2**, luînd valori în intervalul `[3,radacina(n)]`.
Numărul exact de treceri prin buclă nu este fix, dar limitat la maximum

`[radacina(n)]-[radacina(n/2)]-1`

(recomandăm o demonstrare teoretică a acestui rezultat).

c. Deşi `rădăcina(n)` nu este în general un număr întreg, prin operaţia de asignare

`p ←--- rădăcina(n),`

în `p` se reţine numai partea întreagă a rădăcinii.

d. Verificare pentru **n ←--- 79**. Avem
pas 5: **p ←----rădăcina(79) ≡ 8; i ←--- 3**
pas 6: **(i≤p) ≡ (3≤8) ≡ adevărat**
(n=n/i*i) ≡ (79=78) ≡ fals
pas 6 (altfel): **i ←--- i+2 ≡ 5**
pas 6: **(i≤p) ≡ (5≤8) ≡ adevărat**
(n=n/i*i) ≡ (79=75) ≡ fals
pas 6 (altfel): **i ←--- i+2 ≡ 7**
pas 6: **(i≤p) ≡ (7≤8) ≡ adevărat**

(n=n/i*i) ≡ (79=77) ≡ fals
pas 6 (altfel): i ←--- i+2 ≡ 9
pas 6: **(i≤p) ≡ (9≤8) ≡ fals**
şi ciclul se încheie: **79 nu este număr prim** .

Algoritmii `prim_1` şi `prim_2` conduceau la acelaşi rezultat, dar după 38 treceri prin buclă (faţă de numai 4 în acest caz).

Cînd se cer toate numerele prime pînă la `n`, algoritmul poate fi reformulat şi mai eficient:

```
prim_4
m,n,k,i,j  întregi
1. citeşte n
2. dacă n≤1 atunci scrie 'Eroare de date'; salt la 1
           █
3. m ←--- rădăcina(n)
tablou tab[m] de întregi
4. tab[1] ←--- 2 ; k ←--- 1 ; i ←--- 1 ; j ←--- 3
5. dacă n=2 atunci salt la 7
           █
6. cît timp j≤n
          execută
           6.1. cît timp tab[i]≤rădăcina(j)
                         execută
                  dacă j=j/tab[i]*tab[i]
                        atunci salt la 6.3
                        altfel i ←--- i+1
                        █
                         █
           6.2.k ←--- k+1 ; tab[k] ←--- j
           6.3. i ←--- 2 ; j ←--- j+2
          █
7. pentru i ←--- 1,k
       scrie tab[i]
   █
8. Stop
```

O primă observaţie care se impune din compararea celor 4 algoritmi este aceea că

mai eficient nu înseamnă **mai simplu**.

Astfel, algoritmul `prim_4` testează dacă un număr `j` este prim împărţindu-l la toate numerele prime de la **3** la `rădăcina(j)`. În caz că nici o împărţire nu se face exact, `j` este prim şi va fi trecut în `tab`. Altfel, cînd se găseşte un divizor prim al lui `j`, această valoare a lui `j` este părăsită.
Prima poziţie a tabloului `tab` este ocupată de singurul număr prim par: **2**.

3.3.3 Ordonarea crescătoare a unui tablou cu o dimensiune

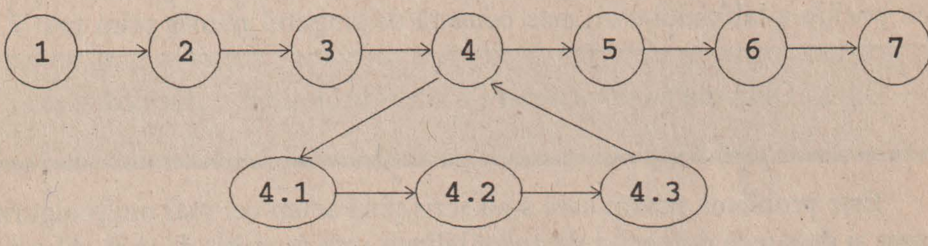
Este problema pentru care s-au scris pînă acum cei mai mulţi algoritmi. Aceasta şi deoarece domeniul de aplicabilitate este deosebit de vast. Algoritmii construiţi sînt în general de tip mixt, utilizînd cicluri cu număr cunoscut sau necunoscut de paşi.

Vom prezenta trei algoritmi diferiţi, fără a avea intenţia de a epuiza posibilităţile relative la această temă .

În cele ce urmează vom presupune că un tablou $\mathbf{a}=(\mathbf{a}_1,\mathbf{a}_2,\ldots,\mathbf{a}_n)$ este ordonat crescător dacă $\mathbf{a}_1\leq\mathbf{a}_2\leq\ldots\leq\mathbf{a}_n$.

```
ord_1
i,n,s  întregi
x real
1. citeşte n
    tablou a[n]  de reale
2. pentru i ←-- 1,n
        citeşte a[i]
   █
3. i ←-- 1; s ←-- 0
4. cît timp i<n
          execută
         4.1. dacă a[i]≤a[i+1] atunci salt la 4.3.
                                █
```

```
            4.2. s ←--- 1;x ←--- a[i];a[i] ←--- a[i+1];
                 a[i+1] ←--- x
              4.3. i ←--- i+1
               █
5. dacă s=1 atunci salt la 3
               █
6 pentru i ←--- 1,n
               scrie a[i]
  █
7. Stop
```



<u>Comentarii:</u> a) Algoritmul `ord_1` conţine două bucle, înafara celor de citire/scriere de la paşii 2 respectiv 6 :

▌ cea de la pasul 4, în care contorul de ciclu este `i`. Bucla este parcursă de `n-1` ori (în cazul limită **n ←--- 0**, de zero ori).

▌ paşii 3-4-5 - ciclu determinat de o testare a lui `s`. Variabila `s` nu este un contor, ci — ca la algoritmul `prim_2` — o variabilă indicator ale cărei valori sînt **0** sau **1**; aici `s` ia valoarea **1** numai în cazul cînd a existat cel puţin o parcurgere a pasului 4.2, deci cînd mai erau elemente neordonate crescător. Ciclul 4 nu mai este parcurs cînd vectorul este ordonat (fapt semnalat de indicator prin `s ←--- 0`).

Deci avem de-a face cu un algoritm mixt.

b) Să parcurgem algoritmul pentru tabloul **a=(3, 5, 2, 3)**, deci **n ←--- 4** .

pas 3: **i ←--- 1 ; s ←--- 0**

pas 4: **(i<n) ≡ (1<4) ≡ adevărat**

4.1: **(a[1]≤a[2]) ≡ (3≤5) ≡ adevărat**

4.3: **i ←--- i+1 ≡ 2**

pas 4: **(i<n) ≡ (2<4) ≡ adevărat**

4.1: **(a[2] ≤ a[3]) ≡ (5 ≤ 2) ≡ fals**

4.2: **s ←--- 1, a[2] ←--- 2, a[3] ←--- 5** (permutare)

4.3: **i ←--- i+1 ≡ 3**

Deci acum tabloul este de forma **(3, 2, 5, 3)**.

pas 4: **(i<n) ≡ (3<4) ≡ adevărat**

4.1: **(a[3]≤a[4]) ≡ (5≤3) ≡ fals**

4.2: **s ←--- 1, a[3] ←--- 3, a[4] ←--- 5** (permutare)

4.3: **i ←--- i+1 ≡ 4**

şi se obţine **a=(3, 2, 3, 5)**.

pas 4: **(i<n) ≡ (4<4) ≡ fals**: închiderea buclei 4.

pas 5: **(s=1) ≡ (1=1) ≡ adevărat**

pas 3: **i ←--- 1, s ←--- 0** şi se reia ciclul pas 4

pas 4: **(i<n) ≡ (1<4) ≡ adevărat**

4.1: **(a[1]≤a[2]) ≡ (3≤2) ≡ fals**

4.2: **s ←--- 1, a[1] ←--- 2, a[2] ←--- 3** (permutare)

4.3: **i ←--- i+1 ≡ 2**

Acum **a=(2, 3, 3, 5)**.

Celelalte parcurgeri prin cele două cicluri nu mai modifică tabloul.

c Schimbarea condiţiei de la 4.1. în `a[i]≥a[i+1]` conduce la ordonarea descrescătoare a elementelor tabloului.

Un algoritm diferit poate fi construit plecînd de la elementul maxim al unui tablou (3.2.3). Ceea ce obţinem va fi însă un algoritm cu număr cunoscut de paşi

```
ord_2
n,i,j  întregi
x real
1. citeşte n
      tablou a[n] de reale
2. pentru i ←--- 1,n
                citeşte a[i]
```

```
      ■
3. j ←--- 1
4. cît timp j<n
        execută
        4.1. i ←--- 1
        4.2. cît timp i<n+1-j
                execută
                4.2.1. dacă a[i]>a[n+1-j]
            atunci x ←--- a[i];a[i] ←--- a[n+1-j];
                                a[n+1-j] ←--- x
            ■

                4.2.2. i ←--- i+1
                ■
        4.3. j ←--- j+1
      ■
5. pentru i ←--- 1,n
           scrie a[i]
  ■
6. Stop
```

Pentru aceleaşi date cu care am verificat algoritmul anterior, **a=(3, 5, 2, 3)** avem:

pas 3: j ←--- 1
pas 4: (j<n) ≡ (1<4) ≡ adevărat
  4.1: i ←--- 1
  4.2: (i<n+1-j) ≡ (1<4) ≡ adevărat
    4.2.1: (a[i]>a[n+1-j]) ≡ (a[1]>a[4]) ≡ (3>3) ≡ fals
    4.2.2: i ←--- i+1 ≡ 2
  4.2: (i<n+1-j) ≡ (2<4) ≡ adevărat
    4.2.1: (a[i]>a[n+1-j]) ≡ (a[2]>a[4]) ≡ (5>3) ≡ adevărat
        a[2] ←--- 3, a[4] ←--- 5 (permutare)
    Noul tablou este **(3, 3, 2, 5).**
    4.2.2: i ←--- i+1 ≡ 3
  4.2: (i< n+1-j) ≡ (3<4) ≡ adevărat
    4.2.1: (a[3]>a[4]) ≡ (2>5) ≡ fals
    4.2.2: i ←--- i+1 ≡ 4

4.2: (i<n+1-j) ≡ (4<4) ≡ fals
4.3: j ←--- j+1 ≡ 2
pas 4: **(j<n) ≡ (2<4) ≡ adevărat**
4.1: i ←--- 1
4.2: (i<n+1-j) ≡ (1<3) ≡ adevărat
4.2.1: (a[i]>a[n+1-j]) ≡ (a[1]>a[3]) ≡ (3>2) ≡ adevărat
a[1] ←--- 2, a[3] ←--- 3 (permutare)
şi se ajunge la **(2, 3, 3, 5).**
4.2.2: i ←--- i+1 ≡ 2
Pînă în final tabloul nu se mai modifică.

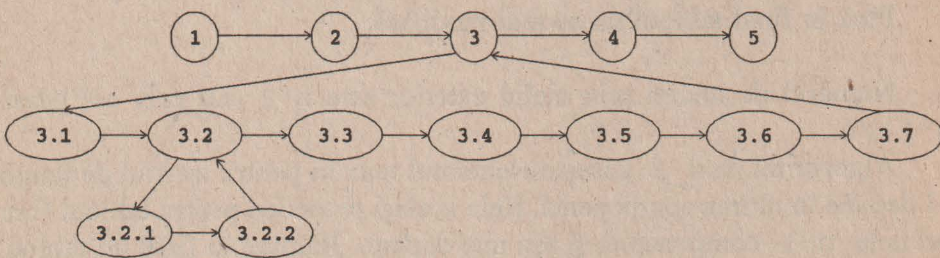
Numărul de treceri prin ciclul exterior este `n-1` iar prin cel interior, `n(n-1)/2`.

Algoritmul `ord_2` găseşte elementul maxim pentru tabloul de lungime `n` şi-l depune în ultima componentă. Reia acelaşi procedeu pentru tabloul format cu primele `n-1` componente, şi aşa mai departe. Rămîne în final un tablou cu o componentă, al cărui element este evident maxim.

În sfîrşit, un al treilea algoritm pentru ordonarea crescătoare a elementelor tabloului este:

```
ord_3
n,i,j,p  întregi
x  real
1. citeşte n
     tablou a[n] de reale
2. i ←--- 2; citeşte a[1]
3. cît timp i≤n
          execută
       3.1. citeşte x; j ←--- 1
       3.2. cît timp j<i
                 execută
            3.2.1. dacă x<a[j] atunci salt la 3.3

            3.2.2. j ←--- j+1
```

```
        3.3. p ←--- i+1
        3.4. p ←--- p-1; a[p] ←--- a[p-1]
        3.5. dacă p>j+1 atunci salt la 3.4.
        3.6. a[j] ←--- x
        3.7. i ←--- i+1
            ■
 4. pentru i ←--- 1,n
         scrie a[i]
   ■
 5. Stop
```



Fie **a=(3, 5, 2, 3).**

pas 2: **i ←--- 2 , a[1] ←--- 3** (citire)

pas 3: **(i≤n) ≡ (2≤4) ≡ adevărat**

3.1: **x ←--- 5** (citire), **j ←--- 1**

3.2: **(j<i) ≡ (1<2) ≡ adevărat**

3.2.1: **(x<a[j]) ≡ (5<3) ≡ fals**

3.2.2: **j ←--- j+1 ≡ 2**

3.2: **(j<i) ≡ (2<2) ≡ fals**

şi bucla 3.2. se închide.

3.3: **p ←--- i+1 ≡ 3**

3.4: **p ←--- p-1 ≡ 2, a[2] ←--- a[1] ≡ 3**

3.5: **(p>j+1) ≡ (2>3) ≡ fals**

3.6: **a[2] ←--- 5**

3.7: **i ←--- i+1 ≡ 3**

pas 3: **(i≤n) ≡ (3≤4) ≡ adevărat**

3.1: **x ←--- 2** (citire), **j ←--- 1**

3.2: **(j<i) ≡ (1<3) ≡ adevărat**

3.2.1: (x<a[j]) ≡ (2<3) ≡ adevărat
3.3: p ←--- i+1 ≡ 4
3.4: p ←--- p-1 ≡ 3, a[3] ←--- a[2] ≡ 5
3.5: (p>j+1) ≡ (3>2) ≡ adevărat
3.4: p ←--- p-1 ≡ 2, a[2] ←--- a[1] ≡ 3
3.5: (p>j+1) ≡ (2>2) ≡ fals
3.6: a[1] ←--- 2
3.7: i ←--- i+1 ≡ 4

În tabloul a se află în acest moment **(2, 3, 5, -)**, deci o ordonare crescătoare.

pas 3: (i≤n) ≡ (4≤4) ≡ adevărat
3.1: x ←--- 3, j ←--- 1
3.2: (j<i) ≡ (1<4) ≡ adevărat
3.2.1: (x<a[j]) ≡ (3<2) ≡ fals
3.2.2: j ←--- j+1 ≡ 2
3.2.1: (x<a[j]) ≡ (3<3) ≡ fals
3.2.2: j ←--- j+1 ≡ 3
3.2.1: (x<a[j]) ≡ (3<5) ≡ adevărat
3.3: p ←--- i+1 ≡ 5
3.4: p ←--- p-1 ≡ 4, a[4] ←--- a[3] ≡ 5
3.5: (p>j+1) ≡ (4>4) ≡ fals
3.6: a[3] ←--- 3
3.7: i ←--- i+1 ≡ 5
pas 3: (i≤n) ≡ (5≤4) ≡ fals
Se scrie tabloul obţinut : **(2, 3, 3, 5)**.

Algoritmul `ord_3` funcţionează în felul următor:

Numerele se introduc pe rînd. După introducerea unei variabile, se parcurge tabloul şi se găseşte poziţia unde trebuie pus acest număr pentru a avea o ordonare crescătoare.

Ciclurile algoritmului sînt:

pas 3: cu contor `i`, parcurs în `n-1` paşi;

pas 3.2: buclă cu număr necunoscut de paşi deşi conţine un contor `j`; ciclul este parcurs de maxim `i-1` ori;

paşii 3.4 - 3.5: cu contor `p`, parcurs în `i-j+1` paşi. Variabila contor `p` descreşte cu pasul **1**.

pas 4: de contor `i`, pentru scriere.

`ord_3` fiind un algoritm dificil, recomandăm studierea lui detaliată, atît prin parcurgerea sa cu alte exemple cît şi descifrarea amănunţită a fiecărui pas (rolul său, importanţa operaţiilor de condiţie, scrierea cu operaţii de decizie alternativă, etc.

EXERCIŢII

1. Se dau numerele întregi **a** şi **b**. Să se scrie o fracţie ireductibilă **x/y** cu proprietatea **x/y=a/b**.

2. Se dă un număr **n**. Să se afle toţi divizorii săi primi.

3. Să se scrie toate numerele pînă la 100 care au 4 divizori primi.

4. Care sînt numerele **n**, $1 \leq n \leq 100$ care au un număr maxim de divizori primi.

5. Se dau două numere întregi **a** şi **b** prime între ele (cmmdc(**a,b**)=1). Să se determine două numere întregi **x, y** astfel încît **a*x+b*y=1**.

6. Se dau două numere întregi **a** şi **b**. Să se găsească două numere întregi **x** şi **y** astfel încît **a*x+b*y=d** unde **d** = cmmdc(**a,b**).

7. Se dă un număr întreg **n**. Să se găsească o bază **p**<10 (dacă există) în care scrierea lui **n** să se facă folosind doar cifrele 0 şi 2.

8. Dintr-un tablou **a** cu o dimensiune să se obţină un tablou **b** în care toate elementele lui **a** apar o singură dată.

9. În aceleaşi condiţii, să se obţină un tablou **b** cu toate elementele lui **a** care se repetă.

10. Se dă un tablou **a** cu două dimensiuni.În el sînt permise doar două operaţii:
- permutarea a două linii între ele (element cu element);
- permutarea a două coloane între ele.

Să se transforme tabloul în aşa fel încît elementele de pe diagonala principală să fie ordonate descrescător.

11. Se dă o propoziţie. Să se găsească litera care se repetă de cele mai multe ori.

12. Se dă o propoziţie avînd cuvintele separate prin spaţiu şi încheiată cu punct.
Să se determine cîte cuvinte are propoziţia.
Cîte din ele sînt formate din două litere ?
Care este cel mai lung cuvînt ?

Fiecare ascultă numai ce înţelege.
(Goethe)

Anexa 1
Scrierea algoritmilor în limbajul BASIC

Programele au fost scrise în varianta Q–BASIC şi verificate pe un calculator IBM PC–286 .

```
 5 REM media_1
10 INPUT a,b
20 x=(a+b)/2
30 PRINT x
40 END
```

```
 5 REM media_2
10 INPUT a,b
20 a=(a+b)/2
30 PRINT a
40 END
```

```
 5 REM rest
10 INPUT a,b
20 r=a-INT(a/b)*b
30 PRINT r
40 END
```

```
 5 REM perm_1
10 INPUT a,b
20 c=a: a=b: b=c
30 PRINT a,b
40 END
```

```
 5 REM perm_2
10 INPUT a,b
20 a=a+b:b=a-b:a=a-b
30 PRINT a,b
40 END
```

```
 5 REM valtr_1
10 INPUT a,b,c
20 GOSUB 40
30 END
40 INPUT x
```

```
 5 REM valtr_2
10 INPUT a,b,c
20 GOSUB 40
30 END
40 INPUT x
```

```
45 p=a*x+b
50 p=p*x+c
60 PRINT p
70 RETURN
```

```
50 p=a*x*x+b*x+c
60 PRINT p
70 RETURN
```

```
 5 REM max_1
10 INPUT a,b
20 if a>b THEN PRINT a ELSE PRINT b
30 END
```

```
 5 REM max_2
10 INPUT a,b
20 x=a
30 IF x<b THEN x=b
40 PRINT x
50 END
```

```
 5 REM max_3
10 INPUT a,b,c
20 IF a<b THEN GOTO 40
30 IF a>c THEN PRINT a ELSE PRINT c: END
40 IF b>c THEN PRINT b ELSE PRINT c: END
```

```
  5 REM max_4
 10 INPUT a,b,c
 20 IF a<b THEN x=b:y=c:GOSUB 100
              ELSE x=a:y=c:GOSUB 100
 30 END
100 IF x>y THEN PRINT x ELSE PRINT y: RETURN
```

```
 5 REM div_1
10 INPUT a,b
20 IF a>=b THEN GOTO 40
30 c=a: a=b: b=c
40 IF b=0 THEN PRINT "impartire la zero": GOTO 60
45 c=a-INT(a/b)*b
50 IF c=0 THEN PRINT a;" este divizibil cu ";b
          ELSE PRINT a;" nu este divizibil cu ";b
60 END
```

```
 5 REM div_2
10 INPUT a,b
20 IF a<b THEN a=a+b: b=a-b: a=a-b
30 IF a*b=0 THEN
          PRINT "Unul din numere este zero": END
40 IF a=INT(a/b)*b
          THEN PRINT a;" este divizibil cu ";b
          ELSE PRINT a;" nu este divizibil cu ";b
50 END
```

```
 5 REM ecuatie_1
10 INPUT a,b,c
20 IF a<>0 THEN GOTO 80
30 IF b<>0 THEN GOTO 60
40 IF c=0 THEN PRINT "Identitate"
          ELSE PRINT "Ecuatie imposibila"
50 END
60 x=-c/b
70 PRINT "Ecuatie de gradul I cu solutia ";x :END
80 delta=b*b-4*a*c
90 IF delta<0 THEN GOTO 140
100 IF delta=0 THEN GOTO 130
110 x1=(-b-SQR(delta))/2/a: x2-(-b+SQR(delta))/2/a
120 PRINT x1;x2 : END
130 x=-b/2/a: PRINT "Solutie dubla";x : END
140 PRINT "Ecuatia nu are radacini reale ": END
```

```
 5 REM ecuatie_2
10 INPUT a,b,c
20 IF a=0 THEN
     IF b=0 THEN
        IF c=0 THEN PRINT "Identitate"
               ELSE PRINT "Ecuatie imposibila"
   ELSE x=-c/b:PRINT "Ec. de gr.I cu solutia "; x
          ELSE delta=b*b-4*a*c:
   IF delta<0 THEN PRINT "Ec. fara rad. reale "
             ELSE
           IF delta=0 THEN x=-b/2/a:
           PRINT "Solutia dubla";x
                    ELSE
      x1=(-b-SQR(delta)/2/a:
      x2=(-b+SQR(delta)/2/a:
      PRINT x1;x2
30 STOP
```

```
 5 REM joc_1
10 PRINT "Alegeti un nume de vietate din lista A."
15 PRINT "A: rata, colibri, ciine, rima, gaina, condor, cal, fluture,
 erete, pinguin, antilopa, sarpe, porumbel, strut, leu, greiere"

20 PRINT "Este pasare ?":INPUT x$
25 IF x$="nu" THEN GOTO 100
30 PRINT "Traieste si la noi in tara ?":INPUT x$
35 IF x$="nu" THEN GOTO 70
40 PRINT "Este domestica ?": INPUT x$
45 IF x$="nu" THEN GOTO 60
50 PRINT "Ii place apa ?": INPUT x$
55 IF x$="da" THEN PRINT "rata" ELSE PRINT "gaina"
57 END
60 PRINT "Este rapitor ?": INPUT x$
65 IF x$="da" THEN PRINT "erete" ELSE PRINT"porumbel"
67 END
70 PRINT "Zboara ?": INPUT x$
75 IF x$="nu" THEN GOTO 90
80 PRINT "Este mica ?": INPUT x$
85 IF x$="da" THEN PRINT "colibri" ELSE PRINT "condor"
```

```
87 END
90 PRINT "Traieste la Pol ?": INPUT x$
95 IF x$="da" THEN PRINT "pinguin" ELSE PRINT "strut"
97 END
100 PRINT "Este animal ?": INPUT x$
105 IF x$="nu" THEN GOTO 140
110 PRINT "Este domestica ?": INPUT x$
115 IF x$="nu" THEN GOTO 130
120 PRINT "Latra ?": INPUT x$
125 IF x$="da" THEN PRINT "ciine" ELSE PRINT "cal"
127 END
130 PRINT "Este ierbivor ?": INPUT x$
135 IF x$="da" THEN PRINT "antilopa" ELSE PRINT "leu"
137 END
140 PRINT "Se tiraste ?": INPUT x$
145 IF x$="nu" THEN GOTO 160
150 PRINT "Musca ?": INPUT x$
155 IF x$="da" THEN PRINT "sarpe" ELSE PRINT "rima"
157 END
160 PRINT "Cinta ?": INPUT x$
165 IF x$="da" THEN PRINT "greiere"ELSE PRINT "fluture"
167 END
```

```
5 REM joc_2
10 PRINT "Alegeti un nume din lista A."
11 PRINT "A: 1.Rata; 2.Colibri; 3. Ciine; 4. Rima; 5.Gaina; 6.Condor;
   7.Cal; 8.Fluture; 9.Erete; 10.Pinguin; 11.Antilopa; 12.Sarpe;
   13.Porumbel; 14.Strut; 15.Leu; 16.Greiere"
15 PRINT "Lista 1: antilopa, erete, fluture, leu, pinguin, porumbel,
          sarpe, strut"
16 PRINT "Lista 2: cal, condor, gaina, leu, porumbel, rima, sarpe,
          strut"
17 PRINT "Lista 3: antilopa, cal, ciine, colibri, condor, leu, pinguin,
          strut"
18 PRINT "Lista 4: antilopa, cal, ciine, erete, gaina, leu, porumbel,
          rata"
20 PRINT "Numele se afla in lista 1 ?": INPUT x$
```

```
25 IF x$="da" THEN i=1 ELSE i=0
30 PRINT "Numele se afla in lista 2 ?": INPUT x$
35 IF x$="da" THEN i=2*i+1 ELSE i=2*i
40 PRINT "Numele se afla in lista 3 ?": INPUT x$
45 IF x$="da" THEN i=2*i+1 ELSE i=2*i
50 PRINT "Numele se afla in lista 4 ?": INPUT x$
55 IF x$="da" THEN i=2*i+1 ELSE i=2*i
60 IF i>0 THEN PRINT "Numele se afla în lista A pe pozitia ";i
        ELSE PRINT "Greiere"
70 END
```

```
 5 REM joc_3
liniile 10—18 sint similare cu joc_2
20 i=0
30 p=1: GOSUB 200
40 p=2: GOSUB 200
50 p=3: GOSUB 200
60 p=4: GOSUB 200
70 IF i=0 THEN i=16
80 PRINT "Numele se afla in lista A pe pozitia ";i
90 END
200 PRINT "Numele este in lista ";p;" ?": INPUT x$
250 IF x$="da" THEN i=2*i+1 ELSE i=2*i
300 RETURN
```

```
 5 REM suma_1
10 INPUT n
20 x=0
30 FOR i=0 TO n
40 x=x+i
50 NEXT i
60 PRINT x
70 END
```

```
 5 REM suma_2
10 INPUT n
20 x=0: i=0
30 WHILE i<=n
40 x=x+i: i=i+1
50 WEND
60 PRINT x
70 END
```

```
 5 REM suma_3
10 INPUT n
20 x=0
30 IF n=0 THEN GOTO 70
40 FOR i=1 TO n
50 x=x+i
60 NEXT i
70 PRINT x
80 END
```

```
 5 REM suma_4
10 INPUT n
20 x=0: i=0
30 DO
40 x=x+i: i=i+1
50 LOOP UNTIL i>n
60 PRINT x
70 END
```

```
 5 REM suma_5
10 INPUT n
20 x=0: i=0
30 x=x+i: i=i+1
40 IF i<=n THEN GOTO 30
          ELSE PRINT x
50 END
```

```
 5 REM suma_6
10 INPUT n
20 x=0
30 FOR i=n TO 0 STEP -1
40 x=x+i
50 NEXT i
60 END
```

```
5    REM suma_7
10   INPUT n
20   x=0
30   FOR i=0 TO n
40   a=x:b=i:GOSUB 100
45   x=a
50   NEXT i
55   PRINT x
60   END
100 a=a+b: RETURN
```

```
 5 REM comp_1
10 PRINT "Dati valoarea de comparat ": INPUT a
15 PRINT "Dati numarul de elemente": INPUT n
20 j=0
25 PRINT "Dati cele ";n;" elemente :"
30 FOR i=1 TO n
40 INPUT x
50 IF x>a THEN j=j+1
60 NEXT i
70 PRINT "Sint ";j;" elemente mai mari decit ";a:
80 END
```

```
 5 REM comp_2
10 PRINT "Dati valoarea de comparat": INPUT a
15 PRINT "Dati numarul de elemente ": INPUT n
20 mare=0:mic=0:egal=0
30 i=1
35 PRINT "Dati cele ";n;" elemente"
40 WHILE i<=n
41 INPUT x
42 IF x>a THEN mare=mare+1
         ELSE IF x<a THEN mic=mic+1
                     ELSE egal=egal+1
43 i=i+1
45 WEND
50 PRINT "Sint ";mic;" elemente mai mici decit ";a
53 PRINT "Sint ";egal;" elemente egale cu ";a
55 PRINT "Sint ";mare;" elemente egale cu ";a
60 END
```

```
 5 REM comp_3
10 PRINT "Dati valoarea de comparat ": INPUT a
15 PRINT "Dati numarul de componente ": INPUT n
18 IF n=0 THEN GOTO 60
20 mare=0: mic=0: egal=0
30 i=1
35 PRINT "Dati cele ";n;" elemente"
40 DO
41 INPUT x
42 IF x>a THEN mare=mare+1
         ELSE IF x<a THEN mic=mic+1
                     ELSE egal=egal+1
43 i=i+1
45 LOOP UNTIL i>n
50 PRINT "Sint";mare;" elemente mai mari decit ";a
53 PRINT "Sint ";egal;" elemente egale cu ";a
55 PRINT "Sint ";mic;" elemente mai mici decit ";a
60 END
```

```
 5 REM comp_4
10 PRINT "Dati valoarea de comparat "; INPUT a
15 PRINT "Dati numarul de elemente "; INPUT n
20 mare=0: mic=0: egal=0
25 IF n=0 THEN GOTO 50
30 i=1
35 PRINT "Dati cele ";n;" elemente"
40 INPUT x
42 IF x>a THEN mare=mare+1
      ELSE IF x<a THEN mic=mic+1
                 ELSE egal=egal+1
43 i=i+1
45 IF i<=n THEN GOTO 40
50 PRINT "Sint";mare;" elemente mai mari decit ";a
53 PRINT "Sint ";mic;" elemente mai mici decit ";a
55 PRINT "Sint ";egal;" elemente egale cu ";a
60 END
```

```
 5 REM maxim_1
10 PRINT "Dati numarul de elemente ": INPUT n
15 IF n=0 THEN PRINT "Nu sint elemente ?": GOTO 60
20 PRINT "Dati cele ";n;" elemente": INPUT max
30 IF n=1 THEN GOTO 50
40 FOR i=1 TO n-1
41 INPUT x
42 IF x>max THEN max=x
45 NEXT i
50 PRINT "Cel mai mare numar este ";max
60 END
```

```
 5 REM maxim_2
10 PRINT "Dati numarul de elemente "; INPUT n
15 IF n=0 THEN GOTO 50
20 max=-1000000
25 PRINT "Dati cele ";n;" elemente"
30 FOR i=1 TO n
```

```
31 INPUT x
32 IF x>max THEN max=x
35 NEXT i
40 PRINT "Cel mai mare element este ";max
50 END
```

```
 5 REM maxmin_1
10 PRINT "Dati numarul de elemente ": INPUT n
15 IF n=0 THEN GOTO 60
20 max=-1000000: min=1000000
30 kmax=0: kmin=0
35 PRINT "Dati cele ";n;" elemente"
40 FOR i=1 TO n
41 INPUT x
42 IF max<x THEN max=x: kmax=1
              ELSE IF max=x THEN kmax=kmax+1
43 IF min>x THEN min=x: kmin=1
              ELSE IF min=x THEN kmin=kmin+1
45 NEXT i
50 PRINT "Cel mai mare numar este ";max;
        ". El apare de ";kmax;" ori"
55 PRINT "Cel mai mic numar este ";min;
        ". El apare de ";kmin;" ori"
60 END
```

```
 5  REM maxmim_2
10  PRINT "Dati numarul de elemente ": INPUT n
15  IF n=0 THEN GOTO 60
20  max=-1000000: min=1000000
30  kmax=0: kmin=0
35  PRINT "Dati cele ";n;" elemente"
40  FOR i=1 TO n
41  INPUT n
42  a=x: b=max: k=kmax: GOSUB 100

43  max=b: kmax=k
44  a=-x: b=-min: k=kmin: GOSUB 100
45  min=-b: kmin=k
47  NEXT i
50  PRINT "Cel mai mare numar este ";max;
```

```
        ". El apare de ";kmax;" ori"
55  PRINT "Cel mai mic numar este ";min;
        ". El apare de ";kmin;" ori"
60  END
100 IF a>b THEN b=a: k=1
        ELSE IF a=b THEN k=k+1
110 RETURN
```

```
 5 REM null(n)
10  PRINT "Dati dimensiunea tabloului": INPUT n
20  IF n=0 THEN GOTO 40
30  GOSUB 100
40  END
100 DIM a(n)
110 FOR i=1 TO n
120 a(i)=0
130 NEXT i
140 RETURN
```

```
5  REM null(n,m)
10  PRINT "Dati dimensiunile tabloului": INPUT n,m
          20  IF n*m=0 THEN GOTO 40
30  GOSUB 100
40  END
100 DIM a(n,m)
110 FOR i=1 TO n
115 FOR j=1 TO m
120 a(i,j)=0
130 NEXT j
135 NEXT i
140 RETURN
```

```
 5 REM comp_5
10 PRINT "Dati numarul de componente": INPUT n
20 IF n=0 THEN GOTO 99
30 DIM a(n)
35 PRINT "Dati cele ";n;" elemente"
40 FOR i=1 TO n
50 INPUT a(i)
60 NEXT i
```

```
65 PRINT "Dati elementul de comparat": INPUT x
70 mare=0: mic=0: egal=0
80 FOR i=1 TO n
81 IF a(i)<x THEN mic=mic+1
            ELSE IF a(i)>x THEN mare=mare+1
                                ELSE egal=egal+1
82 NEXT i
90 PRINT "Sint";mare;" elemente mai mari decit ";x
92 PRINT "Sint ";mic;" elemente mai mici decit ";x
94 PRINT "Sint ";egal;" elemente egale cu ";x
99 END
```

```
 5  REM maxmin_3
10  PRINT "Dati numarul de componente": INPUT n
20  IF n=0 THEN GOTO 100
30  DIM x(n)
35  PRINT "Dati cele ";n;" elemente"
40  max=-1000000: min=1000000
42  kmax=0: kmin=0
45  FOR i=1 TO n
50  INPUT x(i)
60  NEXT i
65  FOR i=1 TO n
70  a=x(i): b=max: k=kmax: GOSUB 110
75  max=b: kmax=k
80  a=-a: b=-min: k=kmin: GOSUB 110
85  min=-b: kmin=k
90  NEXT i
95  PRINT "Cel mai mare numar este ";max;
          ". El apare de ";kmax;" ori."
97  PRINT "Cel mai mic numar este ";min;

          ".El apare de ";kmin;" ori."
100 END
110 IF a>b THEN b=a: k=1
          ELSE IF a=b THEN k=k+1
120 RETURN
```

```
 5  REM maxmin_4
10  PRINT "Dati numarul de componente": INPUT n
20  IF n=0 THEN GOTO 110
30  DIM a(n),maxim(n),minim(n)
35  PRINT "Dati cele ";n;" elemente"
40  FOR i=1 TO n
45  INPUT a(i)
50  NEXT i
55  kmax=1: kmin=1: maxim(1)=1: minim(1)=1
60  max=(a(1): min=a(1): i=2
70  WHILE i<=n
71  IF a(i)>max THEN kmax=1: max=a(i): maxim(1)=i
              ELSE IF a(i)=max THEN kmax=kmax+1:
                                    maxim(kmax)=i
72  IF a(i)<min THEN kmin=1: min=a(i): minim(1)=i
              ELSE IF a(i)=min THEN kmin=kmin+1:
                                    minim(kmin)=i
75  i=i+1
80  WEND
85  PRINT "Elementul maxim este ";max;
          ". El se afla in tablou pe pozitiile ";
88  FOR i=1 TO kmax
90  PRINT maxim(i);",";
92  NEXT i
95  PRINT "Elementul minim este ";min;
          ". El se afla in tablou pe pozitiile ";
90  FOR i=1 TO kmin
100 PRINT minim(i);",";
105 NEXT i
110 END
```

```
 5 REM euclid_1
10 PRINT "Dati cele doua numere": INPUT a,b
20 IF a*b=0 THEN PRINT "Eroare de date": GOTO 80
30 IF a<b THEN r=a: a=b: b=r
40 IF a=b THEN GOTO 70
50 r=a-INT(a/b)*b
60 IF r<>0 THEN a=b: b=r: GOTO 50
70 PRINT "Cel mai mare divizor comun este ";b
80 END
```

```
 5 REM euclid_2
10 PRINT "Dati cele doua numere": INPUT a,b
15 IF a*b=0 THEN PRINT "Eroare de date": GOTO 50
20 WHILE b<>0
25 a=a+b-INT(a/b)*b
30 b=a-b: a=a-b
35 WEND
40 PRINT "Cel mai mare divizor comun este ";a
50 END
```

```
 5 REM euclid_3
10 PRINT "Dati cele doua numere":INPUT a,b
15 IF a*b=0 THEN PRINT "Eroare de date": GOTO 50
18 PRINT "Cel mai mare divizor comun este ";
20 WHILE a>b
25 a=a-b
35 WEND
40 IF a=b THEN PRINT b: GOTO 50
         ELSE x=a: a=b: b=x: GOTO 20
50 END
```

```
 5 REM euclid_4
10 PRINT "Dati cele doua numere": INPUT a,b
15 IF a*b=0 THEN PRINT 'Eroare de date": GOTO 50
18 PRINT 'Cel mai mare divizor comun este ";
20 WHILE a<>b
25 IF a<b THEN b=b-a ELSE a=a-b
35 WEND
40 PRINT b
50 END
```

```
 5 REM prim_1
10 PRINT "dati numarul"
12 INPUT n
15 IF n<=1 THEN PRINT "Eroare de date.Dati alt
                        numar": GOTO 12
20 i=2
30 WHILE i<=n/2
40 IF n=INT(n/i)*i THEN GOTO 70
                  ELSE i=i+1
```

```
50 WEND
60 PRINT n;" este numar prim": END
70 PRINT n;" nu este numar prim": END
```

```
 5 REM prim_2
10 PRINT "Dati numarul"
12 INPUT n
15 IF n<=1 THEN PRINT "Eroare de date.Dati alt
                                  numar": GOTO 12
17 p=1
20 IF n=2 THENN GOTO 60
30 FOR i=2 TO n/2
40 IF n=INT(n/i)*i THEN p=0
50 NEXT i
60 IF p=1 THEN PRINT n;" este numar prim"
          ELSE PRINT n;" nu este numar prim"
70 END
```

```
 5 REM prim_3
10 PRINT "Dati numarul ": INPUT n
20 IF n<=1 THEN PRINT "Eroare de date.": GOTO 10
30 IF n<=3 THEN GOTO 70
40 IF n=INT(n/2)*2 THEN GOTO 80
50 p=SQR(n): i=3
60 WHILE i<=p
65 IF n=INT(n/i)*i THEN GOTO 80
                    ELSE i=i+2
67 WEND
70 PRINT n;" este numar prim.":END
80 PRINT n;" nu este numar prim.": END
```

```
 5 REM prim_4
10 PRINT "Dati numarul ": INPUT n
20 IF n<=1 THEN PRINT "Eroare de date.": GOTO 10
30 DIM tablou(n)
40 tablou(1)=2: k=1: i=1: j=3
50 IF n=2 THEN GOTO 70
```

```
60 WHILE j<=n
61 WHILE tablou(i)<=SQR(j)
65 IF j=INT(j/tablou(i))*tablou(i) THEN GOTO 68
                                     ELSE i=i+1
66 WEND
67 k=k+1: tablou(k)=j
68 i=2: j=j+2
69 WEND
70 FOR i=1 TO k-1
75 PRINT tablou(i);",";
77 NEXT i
78 PRINT tablou(k);"."
80 END
```

```
 5 REM ord_1
10 PRINT "Dati dimensiunea tabloului ": INPUT n
15 DIM a(n)
20 IF n<=0 THEN PRINT "Eroare de date": GOTO 10
23 PRINT "Dati elementele tabloului"
25 FOR i=1 TO n
30 INPUT a(i)
35 NEXT i
37 i=1: s=0
40 WHILE i<n
41 IF a(i)<=a(i+1) THEN GOTO 43
42 s=1: x=a(i): a(i)=a(i+1): a(i+1)=a(i)
43 i=i+1
48 WEND
50 IF s=1 THEN GOTO 37
60 FOR i=1 TO n-1
65 PRINT a(i);",";
70 NEXT i
75 PRINT a(n);"."
80 END
```

```
 5 REM ord_2
10 PRINT "Dati dimensiunea tabloului": INPUT n
15 DIM a(n)
20 IF n<=0 THEN PRINT "Eroare de date": GOTO 10
23 PRINT "Dati elementele tabloului"
25 FOR i=1 TO n
30 INPUT a(i)
35 NEXT i
38 j=1
40 WHILE j<n
41 i=1
42 WHILE i<n+1-j
45 IF a(i)>a(n+1-j) THEN
               x=a(i): a(i)=a(n+1-j): a(n+1-j)=x
47 i=i+1
50 WEND
55 j=j+1
60 WEND
65 PRINT "Tabloul ordonat este:"
70 FOR i=1 TO n-1
75 PRINT a(i);",";
80 NEXT i
85 PRINT a(n);"."
90 END
```

```
 5  REM ord_3
10  PRINT "Dati dimensiunea tabloului ": INPUT n
15  DIM a(n)
20  IF n<=0 THEN PRINT "Eroare de date": GOTO 10
23  PRINT "Dati elementele tabloului"
25  i=2: INPUT a(1)
30  WHILE i<=n
35  INPUT x: j=1
40  WHILE j<i
45  IF x<a(j) THEN GOTO 60
47  j=j+1
50  WEND
60  p=i+1
```

```
65  p=p-1: a(p)=a(p-1)
70  IF p>j+1 THEN GOTO 65
75  a(j)=x
80  i=i+1
85  WEND
87  PRINT "Tabloul ordonat este :"
90  FOR i=1 TO n-1
95  PRINT a(i);",";
100 NEXT i
105 PRINT a(n);"."
110 END
```

Oamenii vorbesc despre ştiinţă;
dar cine crede tot ce vorbeşte ?
(Gr. C. Moisil)

Anexa 2
Scrierea algoritmilor în limbajul PASCAL

Observaţie: Programele au fost scrise în varianta PASCAL 5.5 şi au fost verificate pe un calculator IBM PC-286.

```
program media_1;
var a,b,x:integer;
begin
  readln(a,b);
  x:=(a+b)/2 ;
  writeln(x);
end.
```

```
program media_2;
var a,b: integer;
begin
  readln(a,b);
  a:=(a+b)/2;
  writeln(a);
end.
```

```
program rest;
var a,b,r:integer;
begin
  readln(a,b); r:=a mod b;
  writeln(r);
end.
```

```
program perm_1;
var a,b,c:real;
begin
  readln(a,b);
  c:=a;a:=b;b:=c;
```

```
program perm_2;
var a,b:real;
begin
  readln(a,b);
  a:=a+b; b:=a-b; a:=a-b;
```

```
  writeln(a,b);                    writeln(a,b);
end.                             end.
```

```
program valtr_1;
var a,b,c:integer;
procedure trinom(a,b,c:integer);
var x,p:integer;
begin
  write('Dati valoarea: ');readln(x);
  p:=a*x+b; p:=p*x+c;
  writeln('Valoarea trinomului este ',p);
end;
begin
  write('Dati coeficientii trinomului ');readln(a,b,c);
  trinom(a,b,c)
end.
```

```
program valtr_2;
procedure trinom(a,b,c:integer);
var x,p:integer;
begin
  write('Dati valoarea: '); readln(x);
  p:=a*x*x+b*x+c;
  writeln('Valoarea trinomului este ',p);
end;
var a,b,c:integer;
begin
  write('Dati coeficientii trinomului ');readln(a,b,c);
  trinom(a,b,c)
end.
```

```
program max_1;
var a,b:integer;
begin
```

```
  write('Dati cele doua numere: '); readln(a,b);
  write('Cel mai mare dintre ele este: ');
  if a>b then writeln(a)
         else writeln(b);
end.
```

```
program max_2;
var a,b,x:integer;
begin
  write('Dati cele doua numere: '); readln(a,b);
  write('Cel mai mare dintre ele este: ');
  x:=a; if x<b then x:=b;
  writeln(x);
end.
```

```
program max_3;
var a,b,c:integer;
begin
  write('Dati cele trei numere: '); readln(a,b,c);
  write('Cel mai mare dintre ele este: ');
  if a<b then
          if b<c then writeln(c)
                 else writeln(b)
         else
          if a<c then writeln(c)
                 else writeln(a)
end.
```

```
program max_4;
procedure max(x,y:integer);
begin
  write('Cel mai mare numar este: ');
  if x<y then writeln(y)
         else writeln(x)
```

```
end;
var a,b,c:integer;
begin
  write('Dati cele trei numere: '); readln(a,b,c);
  if a<b then max(b,c) else max(a,c)
end.
```

```
program div_1;
var a,b,c:integer;
begin
  write('Dati cele doua numere: '); readln(a,b);
  if a<b then begin
              c:=a;a:=b;b:=c
             end;
  if b=0 then writeln('Impartire la zero')
        else begin
              c:=a mod b;
              if c=0 then writeln(a,' este divizibil cu ',b)
                     else writeln(a,' nu este divizibil cu ',b)
             end;
end.
```

```
program div_2;
var a,b:integer;
begin
  write('Dati cele doua numere: '); readln(a,b);
  if a<b then begin
               a:=a+b;b:=a-b;a:=a-b;
             end;
  if a*b=0 then writeln('Unul din numere este zero')
          else
           if (a mod b)=0 then writeln(a,' este divizibil cu ',b)
                          else writeln(a,' nu este divizibil cu ',b);
end.
```

```
program ecuatie_1;
var a,b,c,delta,x1,x2: real;
label e1,e2,e3,e4;
begin
  write('Dati coeficientii ecuatiei: '); readln(a,b,c);
  if a<>0 then goto e1;

  if b<>0 then goto e2;
  if c=0 then writeln('Identitate')
        else writeln('Ecuatie imposibila');
  goto e3;
 e2: x1:=-c/b;
    writeln('Ecuatie de gradul 1 cu solutia ',x1);
    goto e3;
 e1: delta:=b*b-4*a*c;
    if delta<0 then begin
                      writeln('Ecuatia nu are radacini reale');
                      goto e3;
                    end;
    if delta=0 then goto e4;
    x1:=(-b-sqrt(delta))/2./a; x2:=(-b+sqrt(delta))/2./a;
    writeln('Ecuatia are doua radacini distincte: ',x1,',',x2);
    goto e3;
 e4: x1:=-b/2./a;
    writeln('Ecuatia are radacina dubla ',x1);
 e3:end.
```

```
program ecuatie_2;
var a,b,c,delta,x1,x2: real;
begin
  write('Dati coeficientii ecuatiei: '); readln(a,b,c);
  if a=0 then
        if b=0 then
                if c=0 then writeln('Identitate')
```

```
                    else writeln('Ecuatie imposibila')
              else begin
                 x1:=-c/b;
                 writeln('Ecuatie de gradul 1 cu solutia ',x1);
                   end
        else begin
              delta:=b*b-4*a*c;

              if delta<0 then begin
                         writeln('Ecuatia nu are radacini reale');
                             end
                         else
                           if delta>0 then begin
                         x1:=(-b-sqrt(delta))/2./a; x2:=(-b+sqrt(delta))/2./a;
                         writeln('Ecuatia are doua radacini distincte: ',x1,',',x2);
                                    end
                                  else begin
                                    x1:=-b/2./a;
                                    writeln('Ecuatia are radacina dubla ',x1);
                                       end
              end
end.
```

```
program joc_1;
var x:string;
begin
  writeln('Alegeti un nume din lista A. El va fi ghicit din patru intrebari');
  writeln('A: rata, colibri, ciine, rima, gaina, condor, cal, fluture, erete,
           pinguin,antilopa, sarpe, strut, leu, greiere');
  write('Este pasare ? '); readln(x);
  if x='da' then begin
       write('Traieste la noi in tara ? '); readln(x);
       if x='da' then begin
                 write('Este domestica ? '); readln(x);
```

```
                 if x='da' then begin
                            write('Ii place apa ? '); readln(x);
                            if x='da' then writeln('Rata')
                                  else writeln('Gaina')
                           end
                       else begin
                            write('Este rapitor ? '); readln(x);
                            if x='da'  then writeln('Erete')

                                   else writeln('Porumbel')
                           end
                end
             else begin
                   write('Zboara ? '); readln(x);
                   if x='da' then begin
                             write('Este mica ? '); readln(x);
                             if x='da' then writeln('Colibri')
                                   else writeln('Condor')
                            end
                        else begin
                             write('Traieste la Pol ? '); readln(x);
                             if x='da'  then writeln('Pinguin')
                                    else writeln('Strut')
                            end
                   end
           end
       else begin
    write('Este animal ? '); readln(x);
    if x='da' then begin
                write('Este domestic ? '); readln(x);
                if x='da' then begin
                           write('Latra ? '); readln(x);
                           if x='da' then writeln('Ciine')
                                 else writeln('Cal')
```

```
                    end
                 else begin
                        write('Este iebivor ? '); readln(x);
                        if x='da' then writeln('Antilopa')
                                  else writeln('Leu')
                     end
         end
      else begin
            write('Se tiraste ?'); readln(x);

            if x='da' then begin
                              write('Musca ? '); readln(x);
                              if x='da' then writeln('Sarpe')
                                        else writeln('Rima')
                           end
                      else begin
                              write('Cinta ?'); readln(x);
                              if x='da'  then writeln('Greiere')
                                         else writeln('Fluture')
                           end
         end
   end
end.
```

```
program joc_2;
var i:integer;
   x:string;
begin
  writeln('Alegeti un nume din lista A. El poate fi ghicit din 4 intrebari.');
  writeln('A: 1.Rata; 2.Colibri; 3.Ciine; 4.Rima; 5.Gaina; 6.Condor; 7.Cal;
           8.Fluture; 9.Erete; 10.Pinguin; 11.Antilopa; 12.Sarpe; 13.Porumbel;
          14.Strut; 15.Leu; 16.Greiere.');
  writeln('Lista 1: antilopa,erete,fluture,leu,pinguin,porumbel,sarpe,strut');
  writeln('Lista 2: cal,condor,gaina,leu,porumbel,rima,sarpe,strut');
```

```
  writeln('Lista 3: antilopa,cal,ciine,colibri,condor,leu,pinguin,strut');
  writeln('Lista 4: antilopa,cal,ciine,erete,gaina,leu,porumbel,rata');
  write('Numele se afla 1n lista 1 ? '); readln(x);
  if x='da' then i:=1
          else i:=0;
  write('Numele se afla in lista 2 ? '); readln(x);
  if x='da' then i:=2*i+1
          else i:=2*i;
  write('Numele se afla in lista 3 ? '); readln(x);
  if x='da' then i:=2*i+1

          else i:=2*i;
  write('Numele se afla in lista 4 ? '); readln(x);
  if x='da' then i:=2*i+1
          else i:=2*i;
  if i>0 then writeln('Numele este in lista A pe pozitia ',i)
        else writeln('Greiere')
end.
```

```
program joc_3;
var p,i:integer;
procedure intrebare(p,j:integer);
var x:string;
begin
  write('Numele se afla in lista ',p,' ? '); readln(x);
  if x='da' then i:=2*j+1
          else i:=2*j;
end;
begin
        — primele 8 linii sînt identice cu cele de la joc_2 —
  i:=0;
  intrebare(1,i);
  intrebare(2,i) ;
  intrebare(3,i);
```

```
  intrebare(4,i);
  if i=0 then i:=16;
  writeln('Numele se afla in lista A pe pozitia ',i)
end.
```

```
program suma_1;
var i,n,suma:integer;
begin
  write('Cite numere sint in suma ? '); readln(n);
  suma:=0;
  for i:=0 to n do

              suma:=suma+i;
  writeln('Suma primelor ',n,' numere naturale este ',suma);
end.
```

```
program suma_2;
var i,n,suma:integer;
begin
  write('Cite numere sint in suma ? '); readln(n);
  suma:=0; i:=0;
  while i<=n do begin
                suma:=suma+i; i:=i+1;
              end;
  writeln('Suma primelor ',n,' numere naturale este ',suma);
end.
```

```
program suma_3;
var i,n,suma:integer;
begin
  write('Cite numere sint in suma ? '); readln(n);
  suma:=0;
  if n>0 then
        for i:=1 to n do suma:=suma+i;
```

```
  writeln('Suma primelor ',n,' numere naturale este ',suma);
end.
```

```
program suma_4;
var i,n,suma:integer;
begin
  write('Cite numere sint in suma ? '); readln(n);
  suma:=0; i:=0;
  repeat  begin suma:=suma+i; i:=i+1; end
  until i>n;
  writeln('Suma primelor ',n,' numere naturale este ',suma);
end.
```

```
program suma_5;
var i,n,suma:integer;
label a;
begin
  write('Cite numere sint in suma ? '); readln(n);
  suma:=0; i:=0;
 a: suma:=suma+i; i:=i+1;
  if i<=n then goto a;
  writeln('Suma primelor ',n,' numere naturale este ',suma);
end.
```

```
program suma_6;
var i,n,suma:integer;
begin
  write('Cite numere sint in suma ? '); readln(n);
  suma:=0;
  for i:=n downto 0  do  suma:=suma+i;
  writeln('Suma primelor ',n,' numere naturale este ',suma);
end.
```

```
program suma_7;
var i,n,suma:integer;
procedure sum(a,b:integer);
begin
   suma:=a+b;
end;
begin
  write('Cite numere sint in suma ? '); readln(n);
  suma:=0;
  for i:=0 to n do  sum(suma,i);
  writeln('Suma primelor ',n,' numere naturale este ',suma);
end.
```

```
program comp_1;
var a,x:real;
    n,i,j:integer;
begin
  write('Dati valoarea de comparat '); readln(a);
  write('Dati numarul de componente '); readln(n);
  j:=0;
  writeln('Dati cele ',n,' componente ');
  for i:=1 to n do begin
                    readln(x); if x>a then j:=j+1;
                   end;
  writeln('Sint ',j,' elemente mai mari decit ',a);
end.
```

```
program comp_2;
var a,x:real;
    n,i,mare,mic,egal:integer;
begin
  write('Dati valoarea de comparat '); readln(a);
  write('Dati numarul de componente '); readln(n);
  mare:=0; mic:=0; egal:=0; i:=1;
```

```
 writeln('Dati cele ',n,' componente ');
 while i<n do  begin
              i:=i+1; readln(x);
              if x>a then mare:=mare+1
                     else if x<a then mic:=mic+1
                                 else egal:=egal+1;
            end;
 writeln('Sint ',mare,' elemente mai mari decit ',a);
 writeln('Sint ',egal,' elemente egale cu ',a);
 writeln('Sint ',mic,' elemente mai mici decit ',a);
end.
```

```
program comp_3;
var a,x:real;
    n,i,mare,mic,egal:integer;
begin
 write('Dati valoarea de comparat '); readln(a);
 write('Dati numarul(nenul) de elemente '); readln(n);
 mare:=0; mic:=0; egal:=0; i:=1;
 writeln('Dati cele ',n,' elemente; ');
 repeat
   read(x);
   if x>a then mare:=mare+1
          else if x<a then mic:=mic+1
                      else egal:=egal+1;
   i:=i+1;
 until i>n;
 writeln(' Sint ',mare,' elemente mai mari decit ',a);
 writeln(' Sint ',egal,' elemente egale cu ',a);
 writeln(' Sint ',mic,' elemente mai mici decit ',a);
end.
```

```
program comp_4;
var a,x:real;
```

```
    n,i,mare,mic,egal:integer;
begin
 write('Dati valoarea de comparat '); readln(a);
 write('Dati numarul de elemente '); readln(n);
 mare:=0; mic:=0; egal:=0;
 writeln('Dati cele ',n,' elemente; ');
 if n>0 then
        for i:=1 to n do begin
                    read(x);
                    if x>a then mare:=mare+1
                          else if x<a then mic:=mic+1
                                   else egal:=egal+1;

                   end;
 writeln(' Sint ',mare,' elemente mai mari decit ',a);
 writeln(' Sint ',egal,' elemente egale cu ',a);
 writeln(' Sint ',mic,' elemente mai mici decit ',a);
end.
```

```
program maxim_1;
var n,i,x,max: integer;
begin
  write('Dati numarul de elemente: '); readln(n);
  if n=0 then writeln('Nu sint elemente.')
       else begin
             writeln('Dati cele ',n,' elemente:'); read(max);
             for i:=1 to n-1 do begin
                              read(x);
                              if x>max then max:=x
                             nd;
             writeln('Cel mai mare numar este: ',max)
                            end
end.
```

```
program maxim_2;
var n,i,x,max: integer;
begin
  write('Dati numarul de elemente: '); readln(n);
  if n=0 then writeln('Nu sint elemente.')
        else begin
               max:=-32200;
               writeln('Dati cele ',n,' elemente:');
               for i:=1 to n do begin read(x); if x>max then max:=x
                                end;
               writeln('Cel mai mare numar este: ',max)
             end
end.
```

```
program maxmin_1;
var n,x,i,max,min,kmax,kmin: integer;
begin
  write('Dati numarul de elemente: '); readln(n);
  if n=0 then writeln('Numar gresit')
        else begin
               max:=-32200; min:=32200;
               kmax:=0; kmin:=0;
               writeln('Dati cele ',n,' elemente:');
               for i:=1 to n do begin
                                  read(x);
                                  if x>max then begin
                                                  max:=x; kmax:=1
                                                end
                                           else if max=x then kmax:=kmax+1;
                                  if min>x then begin
                                                  min:=x; kmin:=1
                                                end
                                          else if min=x then kmin:=kmin+1;
                                end;
```

```
    writeln('Cel mai mare numar este ',max,'. El apate de ',kmax,' ori');
    writeln('Cel mai mic numar este ',min,'. El apare de ',kmin,' ori');
          end
end.
```

```
program maxmin_2;
var n,i,x,max,min,kmax,kmin: integer;
procedure compara(a,b,c,p:integer);
begin
  if p*a>p*b then
        if p>0 then begin
                    max:=a; kmax:=1
                   end
               else begin

                    min:=a; kmin:=1
                   end
        else if a=b then
             if p>0 then kmax:=c+1
                    else kmin:=c+1;
end;
begin
  write('Dati numarul de elemente: '); readln(n);
  if n=0 then writeln('Nu sint elemente de comparat')
        else begin
               max:=-32200;min:=32200; kmax:=0; kmin:=0;
               writeln('Dati cele ',n,' elemente:');
               for i:=1 to n do begin
                  read(x); compara(x,max,kmax,1); compara(x,min,kmin,-1);
                         end;
           writeln('Cel mai mare numar este ',max,'. El apare de ',kmax,' ori');
           writeln('Cel mai mic numar este ',min,'. El apare de ',kmin,' ori');
             end
end.
```

```
program null(n);
var a: array [1..100] of integer;
    i,n:integer;
begin
  write('Dati dimensiunea tabloului '); readln(n);
  if (n<=0) or (n>100) then writeln('Eroare de date')
                    else
                      for i:=1 to n do
                                  a[i]:=0;
  writeln('Primele ',n,' elemente ale tabloului au fost facute zero.')
end.
```

```
program null(m,n);
var a: array [1..90,1..100] of integer;
    i,j,n,m:integer;
begin
  write('Dati dimensiunile tabloului '); readln(m,n);
  if (n<=0) or (n>100) or (m<=0) or (m>90) then writeln('Eroare de date')
                                       else begin
                                            for i:=1 to m do
                                                    for j:=1 to n do a[i,j]:=0;
        writeln('Tabloul format cu primele ',m,' linii si primele ',n,' coloane
                  a fost facut zero.')
                                            end
end.
```

```
program comp_5;
var i,n,mare,egal,mic,x:integer;
    a:array[1..100] of integer;
begin
  write('Dati numarul de componente: '); readln(n);
  if (n<=0) or (n>100) then writeln('Numar gresit.')
        else  begin
                writeln(' Dati cele ',n,' componente:');
```

```
            for i:=1 to n do read(a[i]);
            write('Dati elementul de comparat: ');readln(x);
            mare:=0; mic:=0; egal:=0;
            for i:=1 to n do
                    if a[i]<x then mic:=mic+1
                             else if a[i]>x then mare:=mare+1
                                            else egal:=egal+1;
            writeln('Sint ',mare,' elemente mai mari decit ',x);
            writeln('Sint ',egal,' elemente egale cu ',x);
            writeln('Sint ',mic,' elemente mai mici decit ',x);
          end
end.
```

```
program maxmin_3;
var n,i,max,min,kmax,kmin: integer;
    a: array[1..100] of integer;
begin
  write('Dati numarul de elemente: '); readln(n);
  if (n<=0) or (n>100) then writeln('Numar gresit')
       else begin
            max:=-32200; min:=32200; kmax:=0; kmin:=0;
            writeln('Dati cele ',n,' elemente:');
            for i:=1 to n do read(a[i]);
            for i:=1 to n do begin
                      if a[i]>max then begin max:=a[i]; kmax:=1 end
                               else
                           if max=a[i] then kmax:=kmax+1;
                      if min>a[i] then begin  min:=a[i]; kmin:=1 end
                               else if min=a[i] then kmin:=kmin+1;
                     end;
        writeln('Cel mai mare numar este ',max,'. El apare de ',kmax,' ori');
          writeln('Cel mai mic numar este ',min,'. El apare de ',kmin,' ori');
         end
end.
```

```
program maxmin_4;
var n,i,max,min,kmax,kmin: integer;
   a,maxim,minim: array[1..100] of integer;
begin
  write('Dati numarul de elemente: '); readln(n);
  if (n<=0) or (n>100) then writeln('Numar gresit')
                  else begin
                writeln('Dati cele ',n,' elemente:');
                for i:=1 to n do read(a[i]); writeln;
                kmax:=1; kmin:=1; maxim[1]:=1;minim[1]:=1;
                max:=a[1]; min:=a[1]; i:=2;
                while i<=n  do begin
                           if a[i]>max then begin
                                        max:=a[i];kmax:=1;maxim[1]:=i
                                       end
                                   else if max=a[i] then begin
                                       kmax:=kmax+1; maxim[kmax]:=i
                                                end;
                           if min>a[i] then begin
                                        min:=a[i]; kmin:=1; minim[1]:=i;
                                       end
                                   else if min=a[i] then begin
                                        kmin:=kmin+1; minim[kmin]:=i
                                                end;
                           i:=i+1;
                          end;
                 writeln('Cel mai mare numar este ',max,
                      '. El apare in tablou pe pozitiile:');
                for i:=1 to kmax do write(maxim[i],' '); writeln;
                writeln('Cel mai mic numar este ',min,
                      '. El apare in tablou pe pozitiile: ');
                for i:=1 to kmin do write(minim[i],' '); writeln;
                     end;
 end.
```

```
program euclid_1;
var a,b,r:integer;
label xx;
begin
  write('Dati cele doua numere '); readln(a,b);
  if a<b then begin
              r:=a; a:=b; b:=r
             end;
  if a*b<=0 then write('Eroare de date')
           else begin
              xx: r:=a mod b;

                 if r<>0 then begin
                               a:=b; b:=r;
                               goto xx
                             end;
                 writeln('Cel mai mare divizor este ',b)
               end
end.
```

```
program euclid_2;
var a,b:integer;
begin
  write('Dati cele doua numere '); readln(a,b);
  if a*b<=0 then write('Eroare de date')
           else begin
                 while b>0 do begin
                              a:=b+(a mod b); b:=a-b; a:=a-b
                            end;
                 writeln('Cel mai mare divizor este ',a)
               end
end.
```

```
program euclid_3;
var a,b,x:integer;
label xx;
begin
  write('Dati cele doua numere '); readln(a,b);
  if a*b<=0 then write('Eroare de date')
           else begin
                  write('Cel mai mare divizor comun este ');
                xx: while a>b do a:=a-b;
                  if a=b then writeln(b)
                         else begin
                                x:=a; a:=b; b:=x; goto xx
                              end

                end
end.
```

```
program euclid_4;
var a,b:integer;
begin
  write('Dati cele doua numere '); readln(a,b);
  if a*b<=0 then write('Eroare de date')
           else begin
                  write('Cel mai mare divizor comun este ');
                  while a<>b do
                           if a<b then b:=b-a
                                  else a:=a-b;
                  writeln(b)
                end
end.
```

```
program prim_1;
var i,n:integer;
label xx,xy;
```

```
begin
  write('Dati numarul '); readln(n);
  if n<=1 then writeln('Eroare de date.')
        else begin
                i:=2;
                while i<=n/2 do
                          if (n mod i)=0 then goto xx
                                   else i:=i+1;
                writeln(n,' este numar prim');goto xy;
               xx: writeln(n,' nu este numar prim')
              end ;
xy:end.
```

```
program prim_2;
var i,n,p:integer;
begin
  write('Dati numarul '); readln(n);
  if n<=1 then writeln('Eroare de date.')
        else begin
               p:=1;
               if n>2 then
                     for i:=2 to (n div 2) do if (n mod i)=0 then p:=0;
               if p=1 then writeln(n,' este numar prim')
                    else writeln(n,' nu este numar prim')
              end ;
end.
```

```
program prim_3;
var i,n,p:integer;
    x:real;
label xy,xx;
```

```
begin
  write('Dati numarul '); readln(n);
  if n<=1 then writeln('Eroare de date.')
        else begin
              if n>3 then begin
                           if (n mod 2)=0 then goto xx;
                            x:=n; p:=trunc(sqrt(x)); i:=3;
                            while i<=p do
                                    if (n mod i)=0 then goto xx else i:=i+2;
                          end;
               writeln(n,' este numar prim'); goto xy;
              xx:writeln(n,' nu este numar prim')
             end ;
xy:end.
```

```
program prim_4;
var i,j,k,n:integer;
   tablou:array[1..30] of integer; x:real;
label xx;
begin
  write('Dati numarul '); readln(n);
  if n<=1 then writeln('Eroare de date.')
        else begin
              tablou[1]:=2; k:=1; i:=1; j:=3;
              if n>2 then begin
                          while j<=n do begin
                                        x:=j;
                                        while tablou[i]<=trunc(sqrt(x)) do
                                        if (j mod tablou[i])=0 then goto xx
                                                               else i:=i+1;
                                        k:=k+1; tablou[k]:=j;
                                       xx: i:=2; j:=j+2;
```

```
                    end
                end;
          for i:=1 to k do write(tablou[i],' ');
        end;
  writeln;
end.
```

```
program ord_1;
label 3;
var i,n,s:integer;
    x:real; a:array[1..100] of real;
begin
   write('Numarul de elemente ? '); readln(n);
   for i:=1 to n do readln(a[i]);
3: i:=1; s:=0;
   while i<n do
          begin

            if a[i]>a[i+1] then begin s:=1; x:=a[i]; a[i]:=a[i+1]; a[i+1]:=x  end;
            i:=i+1;
          end;
   if s=1 then goto 3;
   for i:=1 to n do write(a[i],' ');
end.
```

```
program ord_2;
var n,i,j:integer;
    x:real; a:array[1..100] of real;
begin
   write('Numarul de elemente ? '); readln(n);
   for i:=1 to n do readln(a[i]); j:=1;
   while j<n do begin
             i:=1;
             while i<n+1-j do begin
```

```
                        if a[i]>a[n+1-j] then begin
                                                    x:=a[i];
                                                    a [ i ] : = a [ i + 1 ] ;
                                                    a[i+1]:=x;
                                              end;
                        i:=i+1;
                       end;
          j:=j+1;
         end;
   for i:=1 to n do write(a[i],' ');
end.
```

```
program ord_3;
label 3;
var n,i,j,p:integer; y:boolean;
    x:real; a:array[1..100] of real;
begin
   write('Numarul de elemente ? '); readln(n);
   i:=2; readln (a[1]);
   while i<=n do begin
              readln(x); j:=1; y:=false;

              while (j<i) and (not y) do
                              if x<a[j] then y:=true else j:=j+1;
              p:=i+1;
             3: p:=p-1; a[p]:=a[p-1];
              if p>j+1 then goto 3;
              a[j]:=x; i:=i+1;
            end;
   for i:=1 to n do write(a[i],' ');
end.
```

Copii, ce secol e în ograda noastră ?
(Boris Pasternak)

Anexa 3

Index alfabetic al algoritmilor prezentaţi

Nume	Algoritm(pag.)	Program Basic	Program Pascal
maxmin_4	121	158	182
media_1	56	146	164
media_2	57	146	164
null(n)	118	156	180
null(n,m)	118	156	180
ord_1	137	161	187
ord_2	139	162	187
ord_3	141	162	188
perm_1	60	146	164
perm_2	61	146	164
prim_1	132	159	184
prim_2	133	160	185
prim_3	134	160	185
prim_4	136	160	186
rest	58	146	164
suma_1	90	151	173
suma_2	94	151	173
suma_3	95	152	173
suma_4	95	152	174
suma_5	96	152	174
suma_6	97	152	174
suma_7	98	152	175
valtr_1	63	146	165
valtr_2	63	146	165

Un om căzut într-o groapă are drept orizont circumferinţa gropii în care a căzut.
(proverb antic)

BIBLIOGRAFIE

1. A.V.Aho, J.E.Hopcroft, J.D.Ullman - *The Design and Analysis of Computer Algorithms*, Addison - Wesley, Mass. 1974.

2. A. Atanasiu - *Bazele Informaticii*, suport de curs pentru anul II, Tipografia Universităţii Bucureşti, 1987.

3. C. Calude - *Teoria algoritmilor*, Tipografia Universităţii Bucureşti, 1987.

4. D.E.Knuth - *The Art of Computer Programming*, vol. I (Fundamental Algorithms), Addison - Wesley, Mass., 1968.

5. E. Horowitz, S. Sahni - *Fundamentals of Computer Algorithms*, Springer Verlag, 1978.

6. G. Moldovan - *Scheme logice şi programe Fortran*, Ed. Did. şi Ped., 1978.

7. I. Tomescu, A. Leu - *Matematica aplicată în tehnica de calcul*, Manual clasa X, Ed. Did. şi Ped., 1980.

8. *Communication of the Association Computing Machinery*, 1959 - 1986.

9. *Gazeta de Informatică*, nr. 1,2 (1991), 1-11 (1992).

Despre carte:

Ea conţine 50 de algoritmi fundamentali, cu programele respective, în limbajele BASIC şi PASCAL.

Cartea vine în sprijinul celor ce doresc să înveţe să programeze. Poate fi un material didactic preţios pentru informatica predată în clasele VIII-XII.

Cum se scrie un algoritm? Simplu

Despre autor:

Dl. **Adrian Atanasiu** este conferenţiar la Facultatea de Matematică (Universitatea Bucureşti). Predă din 1977 cursuri de scriere a compilatoarelor, bazele informaticii, coduri şi criptografie.

Este redactor al Gazetei de Informatică, membru în juriul mai multor concursuri de informatică pentru elevi.

ISBN 973-95626-6-3

Lei 950