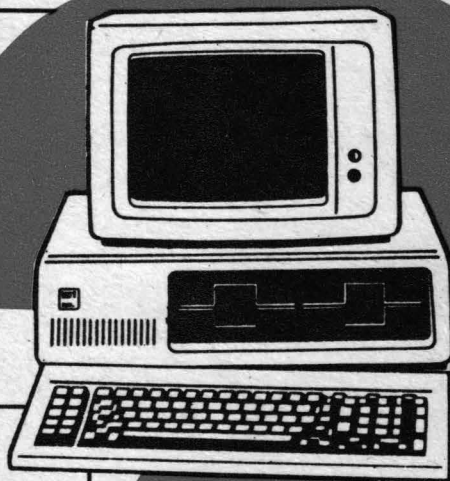
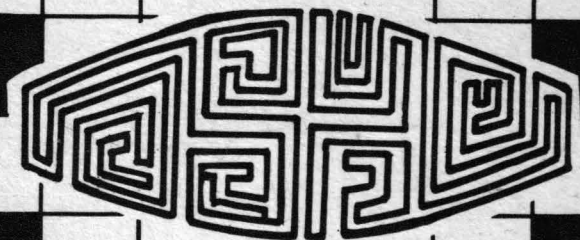


ADRIAN ATANASIU

Cum se scrie un ALGORITHM
SIMPLU!



Editura
Agni

Biblioteca

de Informatică

Adrian Atanasiu

**Cum se scrie un algoritm?
Simplu**

Editura *Agni*

București 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

© Toate drepturile sînt rezervate Editurii AGNI.

Editura AGNI,
CP:30-107, BUCUREŞTI

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

*Nu luați nimic de bun apriori
dacă puteți verifica :*

(R. Kipling)

CUVÎNT ÎNAINTE

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

- La ce moment să fie introduse în programa școlară ore de informatică;
- Ce tip de conținut să aibă aceste ore ?

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

Există însă și un conținut creativ: învățarea de limbaje de programare prin intermediul căroră 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.

- Experiența demonstrează că întîlnirea cu calculatorul este benefică de la o vîrstă cît mai fragedă. Copiii nu au prejudecăți, mintea lor este permanent iscoditoare în căutare de noutăți; 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 funcție de dotare și de profesor, întîlnirea cu calculatorul trebuie să se facă cît mai devreme posibil. Ținîndu-se însă cont de discernămintul copiilor recomandăm ca orele de informatică cu o programă coerentă să se introducă cam prin clasa V.

b. Dacă dotarea spațiului de învățămînt 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 învățare, mai atractiv și mai eficient :

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

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

Lucrarea de față a fost concepută ca un proiect de manual, fiind doar o componentă dintr-un ciclu mai larg. Astfel, în intenția noastră, o carte de informatică de clasa V-a trebuie să conțină probleme simple de logică (sub formă de jocuri) precum și o prezentare a calculatorului; în clasele VI-VII se pot învăța două limbaje de programare scrise special pentru asemenea vîrstă (LOGO și BASIC) iar pentru clasa VIII să fie prevăzut un studiu teoretic elementar despre conținutul 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 așteptarea de propuneri și/sau critici. Să nu uităm totuși că scopul urmărit este cel prezentat la început anume acela că **învățămîntul informatic trebuie să existe.**

Conf. dr. Adrian Atanasiu

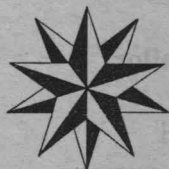
*Adevărul nu se află la capătul unui drum,
ci la acela care deschide o cale .*

(Gelu Negrea Gheorghe)

CUPRINS

Introducere	3
Cuprins	5
I. Algoritmi - prezentare	
1. Introducere	7
2. Proprietăți generale	15
3. Pas de algoritm	20
4. Subalgoritm	24
II. Limbajul algoritmic pseudocod	
1. Generalități	25
2. Comentarii	28
3. Declarații	28
4. Operația de oprire	31
5. Operații de intrare/ieșire	32
6. Expresii aritmetice	36
7. Operația de atribuire (asignare)	41
8. Expresii logice	45
9. Operații de decizie	49
10. Operația de salt fără decizie	53
III. Clasificarea algoritmilor	
1. Algoritmi liniari	56
1.1. Media aritmetică a două numere	56

1.2. Restul împărțirii a două numere întregi	58
1.3. Permutarea a două elemente	60
1.4. Calculul valorii polinomului $p(x)=ax^2+bx+c$	63
2. Algoritmi cu ramificații	67
2.1. Aflarea maximumului între două sau trei numere	67
2.2. Testarea dacă două numere întregi se divid unul cu altul	73
2.3. Rezolvarea ecuației de grad cel mult doi	76
2.4. Algoritm cu ramificații pentru o problemă de căutare	79
3. Algoritmi ciclici	87
3.1. Algoritmi ciclici cu număr cunoscut de pași	90
3.1.1. Suma numerelor naturale pînă la n	90
3.1.2. Cîte elemente dintr-un șir sînt mai mari decît o valoare dată	99
3.1.3. Aflarea maximumului (minimumului) dintre n numere	105
3.2. Variabile de tip tablou	115
3.2.1. Anularea elementelor unui tablou	118
3.2.2. Cîte elemente dintr-un șir sînt mai mari/mici decît x	119
3.2.3. Maximumul/minimumul dintre n numere	120
3.3. Algoritmi ciclici cu număr necunoscut de pași	126
3.3.1. Algoritmul lui Euclid	126
3.3.2. Testarea dacă un număr este prim	132
3.3.3. Ordonarea crescătoare a unui tablou cu o dimensiune	137
Anexa 1: Scrierea algoritmilor în limbajul BASIC	146
Anexa 2: Scrierea algoritmilor în limbajul PASCAL	164
Anexa 3: Index alfabetic al algoritmilor	189
Bibliografie	191



II ALGORITMI - PREZENTARE

II.1 Introducere





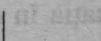


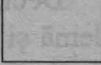




Să presupunem că avem de rezolvat următoarea problemă:

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

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

Fie **A** vasul de 3 litri și **B** - cel de 5 litri.

Soluția este dată de următoarele acțiuni:

- | | A | B | |
|--|---|--|---|
| a. Se umple vasul B cu apă; |  |  | a |
| b. Cu apa din vasul B se umple vasul A ;
Astfel, în A avem 3 l iar în B - 2 litri. |  |  | b |
| c. Se aruncă apa din vasul A ; |  |  | c |
| d. Se pune tot restul de apă din B în A ;
Deci A conține 2 litri de apă iar B este gol. |  |  | d |
| e. Se umple vasul B cu apă; |  |  | e |
| 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 soluția problemei. |  |  | f |

Să rescriem soluția, puțin mai formalizat:

Vom nota $A \leftarrow \text{---}$, $B \leftarrow \text{---}$ umplerea vaselor,
 $\leftarrow \text{---} A$, $\leftarrow \text{---} B$ golirea lor,
 $A \leftarrow \text{---}^n \text{---} B$ turnarea de n litri de apă din B în A .

Deci

a. $B \leftarrow \text{---}$

b. $A \leftarrow \text{---}^3 \text{---} B$

c. $\leftarrow \text{---} A$

d. $A \leftarrow \text{---}^2 \text{---} B$

e. $B \leftarrow \text{---}$

f. $A \leftarrow \text{---}^1 \text{---} B$

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

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

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

**METODA PRIN CARE SE REZOLVĂ O PROBLEMĂ SE
 NUMEȘTE ALGORITM.**

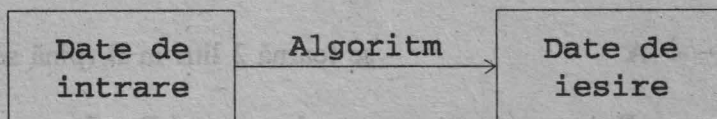
Din problema de sus se desprind două informații:

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

Cele două informații sînt complet deosebite.

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

Putem reprezenta schematic această situație astfel:



Un utilizator trebuie să învețe cum să introducă datele de intrare și cum să citească rezultatul. "Cutie neagră"-dreptunghiul în care datele inițiale 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 și operațiile care trebuiesc efectuate; calculatorul afișează rezultatul pe ecran. Cum ajunge la acest rezultat, nu știm. Ceea ce vrem este ca el să fie obținut

- rapid
- corect

Acea cutie neagră devine însă interesantă în două situații:

- Cînd nu există la dispoziție un algoritm pentru rezolvarea problemei (și 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 greșit, 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 și o altă metodă de rezolvare:

- a. $A \leftarrow \text{---}$ - se umple vasul A (de 3 litri)
- b. $B \leftarrow \text{---} \text{---} A$ - se toarnă 3 litri în B
- c. $A \leftarrow \text{---}$ - se umple din nou vasul A
- d. $B \leftarrow \text{---} \text{---} A$ - se toarnă 2 litri în B (până se umple)
- e. $\leftarrow \text{---} B$ - se golește vasul B
- f. $B \leftarrow \text{---} \text{---} A$ - se golește vasul A în B (cu un litru)
- g. $A \leftarrow \text{---}$ - vasul A se umple din nou
- h. $B \leftarrow \text{---} \text{---} A$ - cei 3 litri din A se toarnă în B (care are acum 4 litri).

Cînd se pot imagina mai mulți algoritmi diferiți 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 pași, pe cînd primul - doar 6), conține mai puține operații și locații, prevede mai multe situații la care să răspundă, etc.

Să mai studiem o problemă:

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

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

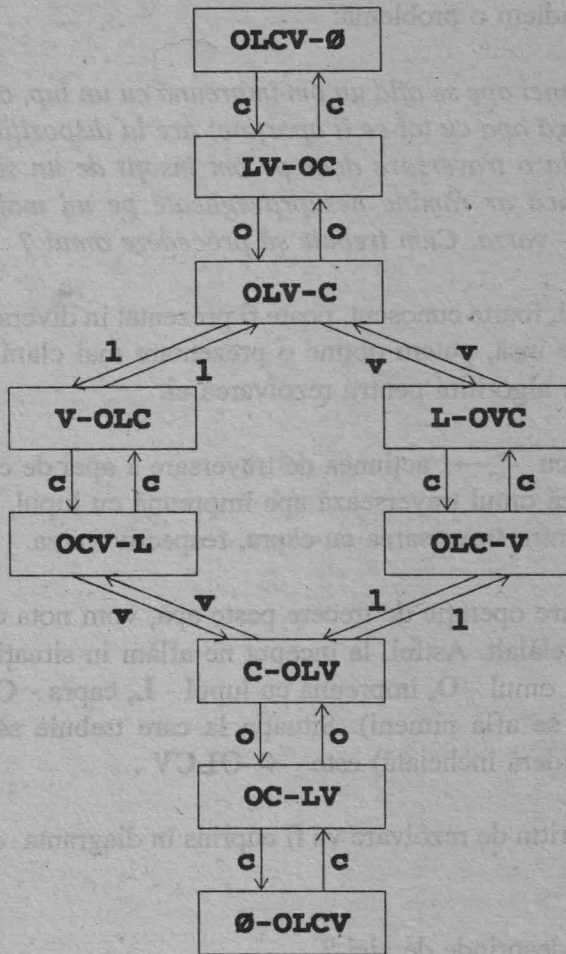
Să notăm cu $-^o \rightarrow$ acțiunea de traversare a apei de către om singur în barcă $-^1 \rightarrow$ dacă omul traversează apa împreună cu lupul, $-^c \rightarrow$,
 $-^v \rightarrow$ similar pentru traversarea cu capra, respectiv varza.

După fiecare operație de trecere peste apă, vom nota cine se află pe un mal și cine - pe celălalt. Astfel, la început ne aflăm în situația $OLCV-\emptyset$ (pe un mal se află omul - **O**, împreună cu lupul - **L**, capra - **C** și varza - **V** ; pe malul celălalt nu se află nimeni). Situația la care trebuie să se ajungă (când problema se consideră încheiată) este $\emptyset-OLCV$.

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

Ce putem desprinde de aici ?

Că problema nu are o singură soluție, ci o infinitate. Fiecare drum care pleacă de sus și ajunge în punctul de jos constituie un algoritm. Astfel omul se poate plimba de pe un mal pe altul de câte ori vrea, ducând sau nu și pe altcineva în barcă, fără ca asta să genereze o soluție greșită (de exemplu, un algoritm ar putea fi - notînd numai acțiunile de traversare: **covclvclvcloc**).



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

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

Deci un algoritm trebuie să completeze în construcția sa anumite lacune (lăsate intenționat sau nu) din enunțul 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 toți 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** - când se știe suportul exact pe care va opera algoritmul.
Această reprezentare este limbajul de programare.

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

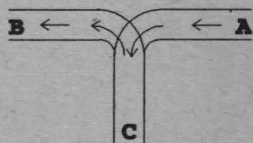
Aceasta este forma finală de reprezentare a unui algoritm, în continuare apărînd doar probleme de utilizare.




EXERCITII

- Să se rezolve prima problemă când avem două vase de 3 și 8 litri pentru a obține : a) 4 litri; b) 5 litri; c) 7 litri;
- Într-un vas mare sînt 12 litri de suc. Cum pot fi împărțiți ei în două părți egale dacă nu avem la dispoziție decît două vase de 5 și 8 litri ? (Deci, în final trebuie să rămînă 6 litri în vasul mare și 6 litri în vasul de 8 litri) .
- Cîți algoritmi se pot da în problema cu traversările dacă sînt permise a) cel mult 11 traversări; b) exact 11 traversări;
- Să presupunem că se fac $2n+1$ traversări. Cîte din acestea le face omul însoțit de capră ?
- 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 mișcări:

- 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 ieșire să fie formată o garnitură cu vagoanele în ordinea 3, 2, 5, 6, 4, 1 ?

Cum ?

1.2 Proprietăți generale

Am definit intuitiv un algoritm ca fiind o metodă folosită pentru rezolvarea unei probleme. Algoritmi există poate de când există civilizația umană. Orice activitate se supune unui algoritm. Pentru a semăna și recolta, pentru a crește vite, pentru a face o casă, trebuie să urmeze anumite reguli.

Denumirea a venit însă destul de târziu; în anul 825, un matematician persan - *Abu Ja'far ibn Musa al Khowarizmi*, a scris o carte de exerciții. Titlul cărții, *Kitab al-jabr wa'l-muqabala*, a introdus în matematică denumirea de **algebră**, iar numele autorului - cea de **algoritm**. Literar, "al Khowarizmi" semnifică "din orașul Khowarizm". Astăzi, acest oraș se numește Khiva și se află în Uzbekistan.

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

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

Semnificația lui a devenit deosebit de importantă în informatică, unde algoritmul reprezintă ceea ce trebuie să facă un calculator pentru rezolvarea unei probleme. Aceasta îl diferențiază 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 ! și se construiește o soluție algoritmică (algoritm).
- b. Nu ! și se demonstrează (destul de dificil) imposibilitatea de a rezolva problema printr-un algoritm.
- c. Nu știm dacă există ! S-ar putea ca problema dată să aibă o soluție algoritmică, și ea se caută.

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

Dar ce proprietăți 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, soluția este simplă. Împărțim cele două numere și găsim $3,(\overline{3})$. Deși sînt o infinitate de zecimale, toate sînt egale, deci răspunsul este clar și obținut în urma aplicării unui algoritm: algoritmul de împărțire, după care se folosește noțiunea de fracție 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 ! Și atunci, cînd poate el da un răspuns la problemă ? Niciodată ! În această situație, 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 unități de timp putem obține 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 pașii 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 împărțirii lui 413 la 27. Dacă însă ea va da restul împărțirii oricăror două numere întregi nenule, atunci avem un algoritm.

TOATE OPERAȚIILE SÎNT DEFINITE ȘI EFECTIVE

O operație 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 diferență există între afirmațiile următoare ?

- ① Formula de rezolvare a ecuației $a \cdot x + b = 0$ este $x = -b/a$.
- ② Ecuația $2 \cdot x - 7 = 0$ are rădăcina reală $x = 7/2$.

În prima situație avem o soluție generală, adevărată pentru orice a și b reale $a \neq 0$. Ea este **utilizată** de algoritm pentru a obține soluția ecuației de gradul I; nu este obținută ca rezultat final.

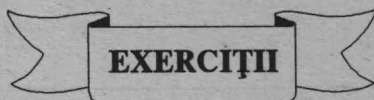
Algoritmul de rezolvare a ecuației de gradul I poate funcționa numai cînd se știu efectiv valorile lui a și b (deci cînd se dă o ecuație particulară). Altfel, operația $-b/a$ care dă soluția acelei ecuații, nu se poate efectua.

Un rezultat algebric înseamnă el însuși 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.

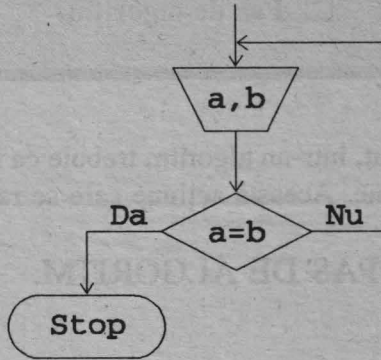


EXERCIIII

1. Se dă o mulțime infinită A . Se poate da un algoritm care să decidă dacă un element a este în mulțimea 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 oraș care au aceeași zi de naștere ?
4. Există în București două persoane care au același număr de fire de păr în cap ? De ce ?
5. Se dă mulțimea

$$M = \{(a, b, c, n) \mid a, b, c, n \in \mathbb{N}, n > 2, a^n + b^n = c^n\}$$
 și $x=0$ dacă $M = \emptyset$, $x=1$ dacă $M \neq \emptyset$.
 Se poate da un algoritm care să-l determine pe x ?
6. Descrieți sub formă de algoritm drumul de acasă pînă la școală.

7. Următoarea schemă logică desemnează sau nu un algoritm ?



8. Considerați că se poate trata drept algoritm:

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

9. Se dă secvența de litere I F M A M I I A

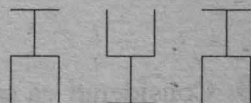
Să se găsească o metodă pentru a determina litera care urmează. Este această metodă un algoritm ?

13 Pas de algoritm

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

PAS DE ALGORITM.

Să studiem următoarea problemă:

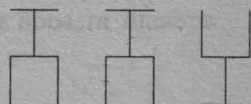


Se dau 3 pahare așezate ca în figură.

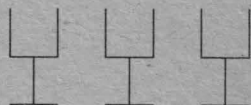
Numim mișcare acțiunea prin care, cu cele două mâini luăm două pahare și le întoarcem simultan. Să se descrie cum pot fi aduse toate cele 3 pahare cu gura în sus, folosind n mișcări(n este un număr cunoscut).

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

- ① se întorc paharele 2 și 3;
Ajungem astfel la configurația



- ② se întorc paharele 1 și 2
și se obține soluția finală.



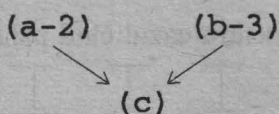
Algoritmul dat mai sus se efectuează în 2 pași de algoritm. Aici, fiecare pas este compus dintr-o singură acțiune (mișcare).

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

Astfel, pentru a obține o mișcare (un pas de algoritm) este necesar ca:

- a**) Să apucăm un pahar cu mâna dreaptă
- b**) Să apucăm alt pahar cu mâna stângă
- c**) Să întoarcem în același timp paharele

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



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

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

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

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

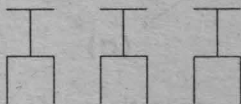
Rezumînd, vom înțelege prin pas de algoritm,

O SECVENȚĂ FINITĂ DE OPERAȚII (ACȚIUNI) CARE SE POT EFECTUA ÎNTR-O UNITATE STABILĂ DE TIMP. UN PAS TREBUIE SĂ CONȚINĂ CEL PUȚIN O ACȚIUNE.

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


EXERCIȚII

1. Să se arate că pentru orice $n > 0$ există un algoritm de rezolvare a problemei anterioare în n pași.
2. Este posibilă problema pentru cazul când paharele sînt așezate inițial



De ce ?

3. Se dă expresia aritmetică $a+b+x*y$. Să se arate ordinea de efectuare a operațiilor în
 - a. 3 pași de algoritm;
 - b. 2 pași de algoritm;
 Se presupune că într-o unitate de timp, o variabilă poate fi prelucrată cel mult odată.

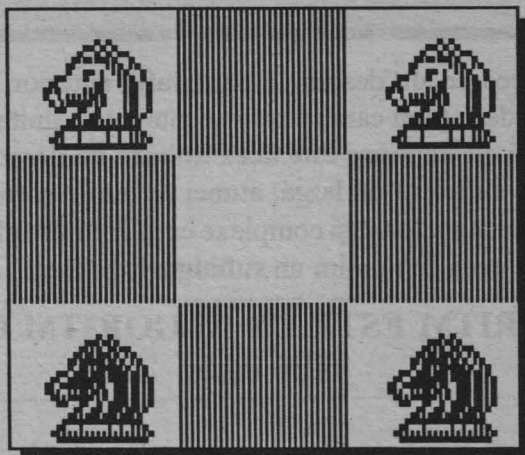
4. Aceeași problemă pentru expresiile aritmetice :

$$(a+b) / (x+y*z) , \quad x = a - 2 / (b+c*x*(a+y+z) - 5);$$
 Care este numărul minim de pași în care se pot rezolva ?

5. Într-un atelier se află 3 muncitori. Într-o unitate de timp:
 - ◆ primul muncitor (a) face un șurub;
 - ◆ al doilea muncitor (b) face o piuliță;
 - ◆ al treilea muncitor (c) le assemblează și le pune într-un plic sigilat împreună cu un preț de vânzare ;

Dimineața nu există nici o piesă făcută, iar la sfîrșitul programului nu rămîn piese dispartate. Să se descrie algoritmul de lucru al celor 3 muncitori.

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



Să se aducă cei doi cai albi pe pozițiile cailor negri și reciproc, folosind numai mutări permise calului în jocul de șah.

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 poziții comune cu primul număr și trei poziții comune cu al doilea număr.

1.4 Subalgoritm

Rezumînd ceea ce am descris în paragraful anterior, un pas de algoritm cuprinde o secvență de acțiuni 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 acțiunile care compun un pas de algoritm sînt destul de numeroase și 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 pași (a , b , c).

Un subalgoritm este foarte util dacă doi sau mai mulți pași de algoritm conțin aceeași secvență de operații, **diferența 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 pași dat anterior, putem construi subalgoritmul **întoarce(x , y)** astfel:

întoarce(x , y)

a se ia paharul x cu mîna dreaptă

b se ia paharul y cu mîna stîngă

c se întorc simultan paharele x și 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 pașii a - b - c în care x este înlocuit cu 2 și y cu 3, iar la pasul 2, aceiași pași sînt parcurși pentru 1 (pe poziția lui x) respectiv 3 (pe poziția lui y).

*Dacă faptele reale nu corespund teoriei,
trebuie să scăpăm de ele.*

(Legea lui Meyer)

III. LIMBAJUL ALGORITMIC PSEUDOCOD

III.1 Generalități

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

- ① Aflarea unui algoritm care să rezolve problema
- ② Transcrierea algoritmului într-un limbaj de programare
- ③ Verificarea corectitudinii rezultatelor programului (validarea algoritmului)

Deosebirea esențială între primii doi pași este modalitatea de reprezentare a algoritmului. Să presupunem că se întâlnesc două persoane care dețin rezolvarea aceleiași probleme, scrisă în limbajele PASCAL și respectiv BASIC. Prima persoană știe numai PASCAL fără a avea nici cea mai mică idee despre programarea în BASIC iar a doua persoană n-a auzit în viața ei de PASCAL, cunoștințele ei de BASIC fiind considerate suficiente. Cum își pot compara cele două persoane programele, să vadă care soluție este mai bună?

O idee ar fi ca una din ele să cedeze și să învețe limbajul cunoscut de partener. Este o metodă coercitivă ! Învingătorul își impune cu forța propriul său limbaj.

O altă rezolvare posibilă este ca cele două persoane interesate să-și stabilească o limbă comună în care să-și transmită algoritmi. Odată stabilit un astfel de limbaj universal - independent de calculatorul folosit - comunicarea

se poate face ușor între orice comunități care folosesc orice limbaj, nu numai PASCAL și BASIC. Se pot stabili contacte și cu utilizatori de LOGO, LISP, DBASE și așa mai departe - fapt remarcabil dacă ținem cont că pe glob se folosesc câteva sute de limbaje de programare.

Aceasta este o metodă diplomatică de înțelegere izvorită din observația că în cele cinci decenii care au trecut de la nașterea limbajelor de programare, nici unul nu a putut cuceri o supremație totală nici în timp și 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-și poată scrie un program în propriul său limbaj. O astfel de reprezentare se întâlnește și în momentul când se învață un limbaj (BASIC de exemplu): este vorba de **schemele logice**.

Ele sînt folosite și acum destul de mult dar cu rețineră din cauza a două deficiențe majore:

- ◆ Într-o schemă logică se dă egală importanță detaliului ca și componentelor principale ale algoritmului. Cum în general detaliile abundă, ele ajung să sufocă algoritmul și 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 însuși). 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 soluțiilor unor probleme formulate într-un anumit limbaj (matematic, natural, etc), permițînd 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 acțiuni numite operații sau instrucțiuni.

În general fiecare pas de algoritm conține o operație.

Cînd într-un pas de algoritm sînt cuprinse mai multe operații, acestea se despart prin **;**.

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



EXERCITII

1. Pentru un număr n dat să se scrie în BASIC și în LOGO programe care să traseze pătrate de latură n . Ce diferențe există în scrierea acestor două programe ? Dar asemănări ?
2. Ce limbaj de programare preferați pentru:
 - ◆ Calculul sumei a două numere
 - ◆ Desenarea unei hărți
3. Ce elemente considerați că trebuie să conțină un limbaj de programare care să poată citi portative cu note și să compună muzică ?

III.2 Comentarii

Un comentariu poate fi plasat oriunde în algoritm și folosește la explicarea operațiilor.

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

```
/* Acesta este un comentariu */
```

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

III.3 Declarații

Toți pașii unui algoritm (deci toate operațiile) prelucrează date. Aceste date sînt în general constante sau variabile.

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

Precizăm : acestea nu sînt toate tipurile posibile de constante; odată cu

creșterea complexității algoritmilor vor fi introduse și altele.

Deci, vom prelucra constante

① **numerice**

② **alfabetice**

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

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

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

Variabilele; așa cum se știe de la limbajele BASIC și LOGO, ele sînt zone de memorie în care se depun constante; aceeași semnificație va fi dată și în cazul **pseudocodului**.

FIECARE VARIABILĂ CONȚINE UN SINGUR TIP DE CONSTANȚE

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 mașinii de scris) sau **logice** (variabile în care se poate păstra una din cele două constante logice definite anterior).

Aceste tipuri se anunță obligatoriu printr-o instrucțiune numită **declarare** care nu aparține 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

anunță că în algoritmul care urmează se va lucra cu variabilele a, b, i în care se rețin valori întregi, x, y - variabile reale și p - o variabilă în care nu se pot plasa decât 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 același algoritm, nici de același tip și nici de tipuri diferite.

Un tip special de declarare este acela al numelui algoritmului.

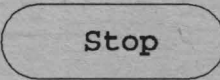
Orice algoritm se identifică (așa cum se identifică persoanele) printr-un nume scris la începutul algoritmului. Numele poate fi urmat eventual și de o listă de variabile numite **parametri**.

Exemplu:

polinom , euclid(a,b) , întoarce(x,y)

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

III.4 Operațiunea de oprire

În schemele logice se notează prin  și marchează sfârșitul algoritmului.

Exemplu:

```

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

```

Dacă ne aflăm într-un subalgoritm și vrem să ne întoarcem în algoritmul care apelează, în loc de Stop se folosește Revenire. Dacă algoritmul se oprește în subalgoritm, aceasta se realizează evident tot cu Stop.

```

algoritm stop1
1. sub      /* se face apel la subalgoritmul sub */
2. Stop    /* algoritmul stop1 se oprește */
sub
1. Revenire /* sub se închide și se revine în stop1 */

```

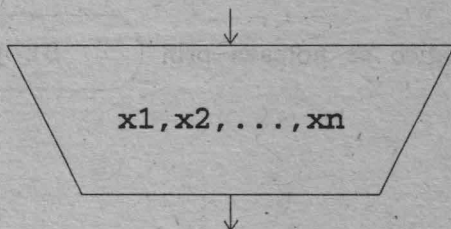
```

algoritm stop2
1. sub     /* se face apel la subalgoritmul sub */
2. Stop   /* operația de oprire */
sub
1. Stop   /* se încheie simultan sub și stop2 */

```

III.5 Operații de intrare/ieșire

Operația de intrare, reprezentată în schema logică prin



este definită

citește x_1, x_2, \dots, x_n

unde x_1, x_2, \dots, x_n sînt nume de variabile.

Prin această operație, în cele n variabile x_i se introduc valori pe care algoritmul le va folosi ulterior.

Se consideră că după acest pas, variabilele x_1, x_2, \dots, x_n sînt **definite**.

Observații:

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

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

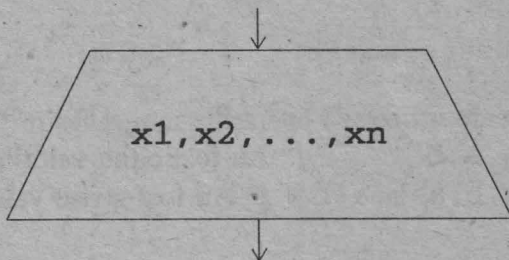
Pentru a elimina eventuale confuzii, o definiție generală se scrie

citește < lista de variabile >

în care **citește** este un cuvînt caracteristic operației de intrare (numit cuvînt cheie), iar < lista de variabile > este o secvență de variabile separate prin virgule (de forma x_1, x_2, \dots, x_n).

- ◆ În lista de variabile există cel puțin o variabilă.
- ◆ Toate variabilele din listă sînt distincte una de alta.

Similar, pentru operația de ieșire avem reprezentarea în schema logică



În pseudocod, definiția este

scrie < lista de variabile >

unde <lista de variabile > are aceeași semnificație de la operația de citire. Aici însă variabilele din listă reprezintă **rezultate** și ele pot fi astfel puse la îndemîna celui care utilizează algoritmul.

De remarcat că pentru ușurința înțelegerii rezultatelor, în această listă pot fi inserate și texte de tip **comentariu** care, spre deosebire de textele **comentariu**, apar la ieșire. Cum ele aparțin însă unui pas de algoritm vor fi încadrate de '.

Exemplu

scrie 'Rezultatul este ', x

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

Rezultatul este 2

Instrucțiunile de intrare/ieșire reprezintă de fapt cele două capete ale schemei de la pagina 9.

Exemplu

Algoritmul următor introduce în variabilele **a** și **b** două numere și scrie (afișează, scoate) aceste numere precum și suma și diferența lor;

algoritm citește/scrie

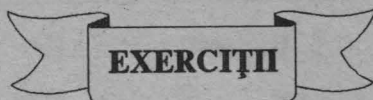
a,b întregi

/* Algoritm care utilizează operațiile intrare/ieșire */

1. citește a,b /* Au fost citite valorile lui **a** și **b** */

2. scrie a,b,a+b,a-b /* Au fost scrise valorile rezultate */

3. Stop



EXERCII

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

2. Cum este bine :

1. citește x
2. scrie 'Valoarea lui x este ',x , sau

1. citește x
2. scrie 'Valoarea lui x este ,x '

3. Este corect algoritmul

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

Dar

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

4. Se dă algoritmul

```
algoritm trecere
a întreg
1. citește 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 operația

i) Stop ii) Revenire

III.6 Expresii aritmetice

În scrierea algoritmilor, deosebit de mult folosite sînt expresiile aritmetice. Utilizarea lor, deși cunoscută, necesită totuși cîteva precauții impuse de restricțiile scrierii pe calculator. Astfel:

a Componentele unei expresii sînt:

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

Orice expresie aritmetică conține cel puțin un operand.

- **operatori binari:** +, -, *, /, ^.

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

Ca o remarcă, exponentierea ^, deși este operator binar, nu există în toate limbajele de programare.

- **operator unar:** -.

Este semnul - care poate apare în fața 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 **funcții** considerate **standard**; alegerea lor este în general dependentă de limbaj. Vom considera totuși cîteva funcții standard folosite universal.

abs definită

$$\text{abs}(x) = x \quad \text{dacă } x \geq 0$$

$$\text{abs}(x) = -x \quad \text{dacă } x < 0$$

int;

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

rădăcina;

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

sin, cos, tan, asin, acos, atan - funcții trigonometrice.

ln - funcție logaritmică, etc.

ORICE OPERATOR SE SCRIE EXPLICIT ÎN EXPRESIE.

O exprimare de forma $a(b+c)$ este greșită. 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 ușoară;
pentru a o stăpîni bine, trebuie să ținem cont de așa numitele priorități de calcul.

c) Priorități de calcul .

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

- ▶ se execută întîi funcțiile standard;
- ▶ urmează prelucrarea expresiilor aritmetice din paranteze;
- ▶ se execută operatorul unar -;
- ▶ se execută exponențierile;
- ▶ se execută înmulțirile și împărțirile;
- ▶ se execută adunările și scăderile.

**DACĂ DOI OPERATORI DE ACEEAȘI PRIORITATE SÎNT
CONSECUTIVI, EI SE EXECUTĂ ÎN ORDINEA APARIȚIEI
LOR - DE LA STÎNGA LA DREAPTA - CU EXCEPȚIA
EXPONENȚIERILOR.**

*Exemple***i. $a+b-c$**

Ordinea de execuție: 1. adunarea lui **a** cu **b**;
2. din rezultatul obținut 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 ținând cont și de ordinea de execuție a operațiilor:

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

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

$$a*(b-x/2*\text{rădăcina}(b)) ?$$

1. rădăcina(**b**);
2. $x/2$;
3. rezultatele de la 1 și 2 se înmulțesc;
4. se scade din **b** rezultatul de la 3;
5. **a** se înmulțește 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 și constantele numerice dintr-o expresie aritmetică se consideră de două tipuri: **întreg și real**.

Datorită faptului că aceste tipuri de operanzi se rețin în calculator în mod diferit, vor diferi și operațiile care se fac cu ei.

Astfel, orice operație aritmetică $a @ b$ se supune următoarei reguli:

DACĂ a ȘI b SÎNT ÎNTREGI, ATUNCI REZULTATUL OPERAȚIEI $a @ b$ ESTE ÎNTREG. DACĂ CEL PUȚIN UNUL DIN OPERANZI ESTE REAL, ATUNCI REZULTATUL ESTE REAL.

Afirmațiile par simple în cazul operatorilor $+$, $-$ și $*$, diferențele nefiind sesizabile la nivel teoretic.

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

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

De exemplu, $5/2$ este întreg; valoarea lui este 2.

De fapt, în această situație are loc egalitatea $a/b = \text{int}(a/b)$.

În schimb

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

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

Deoarece în calculator mulțimile 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 depășirea acestor intervale.

Exemplu

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

Cele mai frecvente astfel de erori pentru variabilele reale sînt cele generate de împărțirea a/b , cînd b ia valori foarte mici; atunci $a/b = a*(1/b)$ și cum $1/b$ devine foarte mare, înmulțirea va genera depășire.

EXERCIIȚII

1. Să se liniarizeze expresiile aritmetice:

$$\frac{a}{2\left(\frac{x+y}{b} + \frac{x+z}{c}\right)} - \frac{b}{a+1.3}$$

$$\frac{\frac{x}{abc} + \frac{y}{a+b+c} + \frac{2}{ab+bc+ac}}{xyz}$$

$$\sqrt{\frac{ab}{\sqrt{c}} - \frac{bc}{\sqrt{a}} + \frac{ac}{\sqrt{b}}}$$

$$3^{\sqrt{a}} * a^{\sqrt{3}}$$

$$\frac{\sqrt{abs(a)}}{a} - \frac{\sqrt{abs(b)}}{b}$$

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

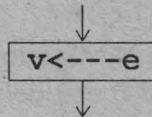
$$\begin{aligned} & \text{rădăcina}(b*b-4*a*c) \\ & (x+y/z/2.5)^(a-abs(a)*c \text{ina}(x*y-43)) \\ & \sin(a*\cos(x+y)+b*\cos(x-y)) \end{aligned}$$

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

$$\begin{array}{cccc} 1/2+1/2+1/2; & 70/3/2; & 104/7/5; & 8/3; \quad 8*1/3; \quad 1/3*8; \\ (5*3-2*4)/2/2; & 3^2^2; & 2^3^4; & 3^2/3; \end{array}$$

III.7 Operația de atribuire (asignare)

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



iar în pseudocod, forma generală este pur și simplu

$$v \leftarrow e$$

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 și 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

$a \leftarrow 2+5$ are ca efect atribuirea valorii 7 variabilei a ;
 $x \leftarrow 'p'$: variabila x a fost declarată de tip caracter și prin această operație, ei i s-a atribuit valoarea 'p'.

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

Observații:

◆ Operația de intrare lucrează similar operației de atribuire. Astfel, în secvența

a real; b caracter ; c logic
 citește a, b, c

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

Dacă am dat

5.2 x adevărat

atunci operația de citire este echivalentă cu secvența de atribuirei

$a \leftarrow 5.2$; $b \leftarrow 'x'$; $c \leftarrow$ adevărat

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

◆ Un algoritm apelează un subalgoritm $\text{sub}(a,b)$; în momentul cînd se ajunge la acest pas, variabilele a și b sînt definite.

Se trece imediat la parcurgerea subalgoritmului sub , al cărui nume este $\text{sub}(x,y)$. Primul lucru care se efectuează în acest moment este secvența de atribuirei

$x \leftarrow a$; $y \leftarrow b$

Exemplu

atribuire

a, b reale

1. citește a /* Se citește în a o valoare reală */

2. $b \leftarrow a$ /* Valoarea lui a se copiază și în b */

3. scrie b /* Se scrie valoarea aflată în b */

4. Stop

Dacă înlocuim pasul 2 cu


2'. b ←--- a+b

operația este greșită deoarece acum variabila **b** din membrul drept nu este definită.

Exemplu

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

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

Atenție  Operația 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 obține astfel o valoare.

☞ În faza a doua, valoarea obținută se depune la adresa variabilei din stînga operației de atribuire (pentru noțiunile de **valoare** și **adresă** se pot revedea orice cărți de prezentare ale limbajelor BASIC, PASCAL sau LOGO).

Exemplu

```
contor
i întreg
1. citește i /* Se dă o valoare lui i */
2. i ←--- i+1 /* Valoarea lui i se mărește 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 același rezultat conduce și algoritmul

```

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

Diferența constă în ceea ce rămâne la sfârșit în variabila i ; valoarea 8 la algoritmul contor, valoarea 7 (nemodificată) la algoritmul increment.

EXERCIȚII

1. Se citește o valoare în variabila a . Să se scrie valoarea lui a fără semn (dacă a ia valoarea -2 , se va scrie 2).

Indicație : $a \leftarrow \dots$ rădăcina ($a \cdot a$)

2. Ce efect au pașii de algoritm :

- | | | |
|--|----|---|
| <p>a) 1. citește a
 2. $a \leftarrow \dots a - a$
 3. $b \leftarrow \dots a$
 4. scrie a, b</p> | și | <p>1. citește a
 2. $b \leftarrow \dots a$
 3. $a \leftarrow \dots a - a$
 4. scrie a, b</p> |
| <p>b) 1. citește a, b
 2. $a \leftarrow \dots a + b$
 3. $b \leftarrow \dots a + b$
 4. scrie a, b</p> | și | <p>1. citește a, b
 2. $b \leftarrow \dots a + b$
 3. $a \leftarrow \dots a + b$
 4. scrie a, b</p> |
| <p>c) 1. citește a
 2. $b \leftarrow \dots a$
 3. $a \leftarrow \dots a + b$</p> | și | <p>1. citește a
 2. $a \leftarrow \dots a + b$
 3. $b \leftarrow \dots a$</p> |

III.5 Expresii logice

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

Sînt două tipuri de astfel de expresii:

a) expresii relaționale

de forma

< expresie aritmetică 1 > < relație > < expresie aritmetică 2 >

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

- < - mai mic
- ≤ - mai mic sau egal
- > - mai mare
- ≥ - mai mare sau egal
- = - egal
- ≠ - diferit

Exemple

$2 < i+1$; $k=3$; $x*x+y*y \geq 0$; rădăcina(p) $\neq 2$.

Efectul unei astfel de expresii relaționale este:

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

Exemplu

Avem expresia relațională $a*b+5 > x$.

Dacă $a \leftarrow 0$, $b \leftarrow 1$, $x \leftarrow 1$, după evaluările celor două expresii aritmetice se obține forma $5 > 1$ care ia valoarea **adevărat**.

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

Expresia

$$a > b > 0$$

nu este o expresie relațională corectă, nici în această formă și nici ca

$$(a > b) > 0 \quad \text{sau} \quad a > (b > 0)$$

Aceasta deoarece - conform construcției a. - o expresie relațională conține un singur operator de tip <relație>.

b expresii logice

sînt expresii construite cu:

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

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

$$\text{nu}(\text{adevărat}) \equiv \text{fals}, \quad \text{nu}(\text{fals}) \equiv \text{adevărat}$$

sau	adevărat	fals
adevărat	adevărat	adevărat
fals	adevărat	fals

și	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 \equiv în loc de $=$; cînd scriem $x \equiv y$ aceasta înseamnă că expresiile x și y (aritmetice sau logice) iau simultan aceleași valori (numerice respectiv logice).

\equiv poartă numele de **identitate**.

Exemple

$(a > b)$ și nu $(x \leq 2)$; aici a, b, x sînt variabile numerice
 a sau b și nu $(c < 0)$; aici a și b sînt variabile logice iar c -
 variabilă numerică.

nu a nu $(x > z)$ este greșită, deoarece al doilea operator nu are
 doi operanzi, ceea ce intră în contradicție cu
 definiția sa de operator unar.

Ca și în cazul expresiilor aritmetice, între operatorii relaționali și logici
 există priorități de calcul. Într-o expresie logică, ordinea de execuție a
 operatorilor este:

- ◆ funcțiile standard
- ◆ expresiile între paranteze
- ◆ expresiile aritmetice (cu prioritățile cunoscute)
- ◆ expresiile relaționale (de la stînga la dreapta)
- ◆ nu
- ◆ și
- ◆ sau.

Exemplu

Fie expresia logică

$a + b * (x - y) > \text{rădăcina}(4)$ și nu $(2 - a = 3^x)$ or v

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

$a \leftarrow 0$, $b \leftarrow 1$, $x \leftarrow 2$, $y \leftarrow 1$, $v \leftarrow \text{fals}$

Ordinea de efectuare a calculelor este :

- ① $\text{rădăcina}(4) \equiv 2$;
- ② $x - y \equiv 2 - 1 \equiv 1$;
- ③ $3^x \equiv 3^2 \equiv 9$;
- ④ $2 - a \equiv 2 - 0 \equiv 2$;
- ⑤ $a + b * (x - y) \equiv 0 + 1 * 1 \equiv 1$;

- ⑥ $2-a=3^x \equiv 2=9 \equiv \text{fals}$;
 ⑦ $a+b*(x-y) > \text{rădăcina}(4) \equiv 1>2 \equiv \text{fals}$;
 ⑧ $\text{nu}(2-a=3^x) \equiv \text{nu}(\text{fals}) \equiv \text{adevărat}$;
 ⑨ $a+b*(x-y) > \text{rădăcina}(4) \text{ și } \text{nu}(2-a=3^x) \equiv \text{fals și adevărat} \equiv \text{fals}$;
 ⑩ $a+b*(x-y) > \text{rădăcina}(4) \text{ și } \text{nu}(2-a=3^x) \text{ or } v \equiv \text{fals or fals} \equiv \text{fals}$.

Deci valoarea întregii expresii este **fals**.

De remarcat diferența între:

- = : operator relațional care compară valorile din membrul stîng și drept și dă ca rezultat o valoare logică: **adevărat** dacă cele două valori sînt egale, **fals** altfel;
 ←--- : operația pseudocod de atribuire;
 ≡ : operația identitate ; expresiile din cei doi membri iau aceleași valori pentru orice definiții ale variabilelor care le compun.

EXERCITII

- Pentru ce x și y expresia $x*x+y*y=0$ ia valoarea **adevărat** ?
Dar $x*x+x+1>0$?
- Să se scrie o expresie logică care să ia valoarea **adevărat** dacă și numai dacă variabila întregă n ia valori între 0 și 120.
- Să se scrie o expresie logică care să ia valoarea **adevărat** dacă și numai dacă variabila a este un număr par mai mare decît -25.
- Se dă expresia logică $\text{nu}(a='a') \text{ și } (a='z') \text{ or } (a='x')$
Cînd este ea adevărată ?
- Se dau două numere reale x și y . Ce expresie relațională ia valoarea **adevărat** dacă și numai dacă:
 ♦ x și y sînt de semne contrare

- ◆ x și y sînt nenule
- ◆ cel puțin unul din ele este 0
- ◆ ambele sînt 0

6. Fie $a \leftarrow 0$, $b \leftarrow 1$, $x \leftarrow 2$, $y \leftarrow$ adevărat

Să se cerceteze dacă următoarele expresii logice sînt corecte și în acest caz să se afle valorile lor:

$$\begin{aligned} a * b > 1 \text{ or } \text{nu}(x > 2 \text{ or } b=1) \\ (a+b+x) > x \text{ și } y \\ (a+b+x) > (x \text{ și } y) \end{aligned}$$

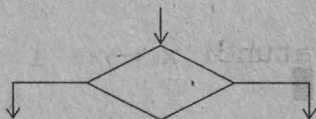
7. Să se arate că au loc identitățile:

$$\begin{aligned} a \text{ și } \text{nu } a &\equiv \text{fals}; \\ a * a \geq 0 &\equiv \text{adevărat}; \\ a \neq b &\equiv \text{nu } (a=b); \\ \text{nu}(a \text{ și } b) &\equiv \text{nu } a \text{ sau } \text{nu } b; \\ \text{nu}(a \text{ sau } b) &\equiv \text{nu } a \text{ și } \text{nu } b; \\ \text{nu}(a > b) &\equiv a \leq b; \\ a < b &\equiv (a \leq b) \text{ și } \text{nu } (a=b); \end{aligned}$$

III.9 Operații de decizie

Folosind operațiile pseudocod definite pînă acum, un algoritm se poate parcurge trecînd pe rînd de la un pas la altul, în ordinea apariției lor. O astfel de parcurgere se numește **secvențială** sau **liniară**.

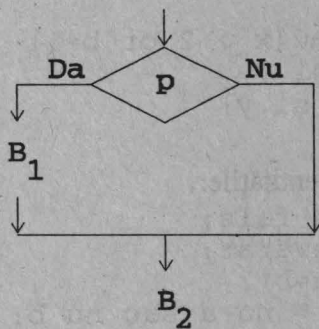
De multe ori însă anumite acțiuni trebuiesc executate numai în urma luării unor hotărîri. Pentru aceasta sînt necesare operații speciale numite **operații de decizie**. În schemele logice o astfel de operație este reprezentată



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

Operații de decizie alternativă

a) pentru varianta



corespunde operația

dacă p atunci B_1

B_2

unde p este o expresie logică, iar B_1 și B_2 - două secvențe de algoritmi formate din unul sau mai mulți pași. Aceasta se numește operație de decizie alternativă **forma scurtă** și prelucrarea ei se face în următoarele etape:

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

Exemplu.

```

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

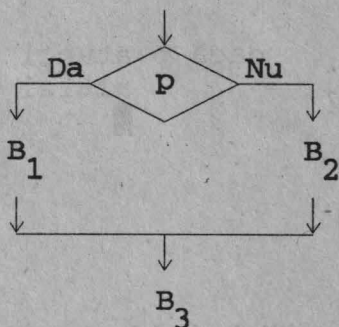
După ce se face testarea $a > 0$ (și se află valoarea acestei expresii relaționale) algoritmul are două ramuri posibile:

$(a > 0) \equiv \text{adevărat}$. Atunci $x \leftarrow 1$ și operația de ieșire dă valoarea 1, sau

$(a > 0) \equiv \text{fals}$. În acest caz x rămîne cu valoarea anterioară și la ieșire va apare valoarea 0.

■ este un marcator care delimitează sfîrșitul secvenței de algoritm B_1 . Lipsa unui astfel de semn distinctiv ar face imposibilă separarea pasului de algoritm cînd se încheie operația de decizie și începe blocul B_2 .

ⓑ Operația generală de decizie alternativă, corespunzătoare schemei logice



este:

dacă p atunci B_1
 altfel B_2

■
 B_3

Prelucrarea se face astfel:

- se evaluează expresia logică p ;
- dacă valoarea lui p este **adevărat**, atunci se execută secvența de algoritmi $B_1 B_3$ (fără B_2);
- dacă valoarea lui p este **fals**, se omite B_1 executîndu-se $B_2 B_3$.

Exemplu

```

dacă a > 0 atunci x ← 1
                altfel x ← 2

```

```

scrie x

```

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

De remarcat că pentru blocul B_1 rolul de delimitator îl joacă cuvintele cheie **atunci** și **altfel**, iar pentru B_2 - cuvântul cheie **altfel** și semnul ■.

Operația de decizie alternativă forma scurtă este un caz particular, când blocul B_2 nu conține nici o operație pseudocod:

```

dacă p atunci B1
                altfel

```

Operații de decizie repetitivă

Sînt introduse în secțiunea referitoare la algoritmii ciclici.

III.10 Operația de salt fără decizie

Nu este o operație esențială, orice algoritm putînd - printr-o aranjare convenabilă a pașilor săi - să nu folosească astfel de instrucțiuni. Nu are echivalent în schemele logice.

Forma generală este

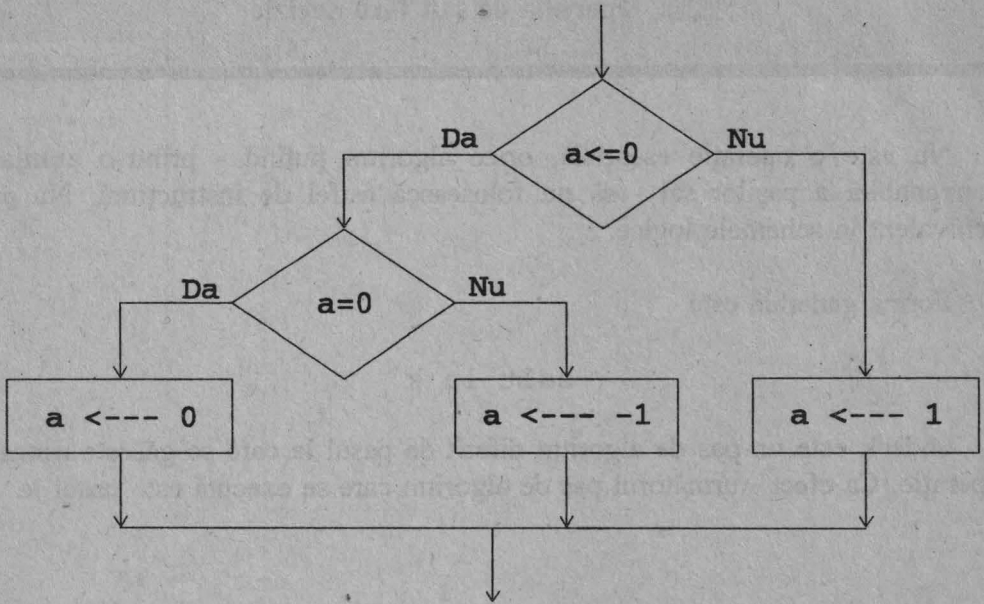
salt la k

unde k este un pas de algoritm diferit de pasul la care se găsește această operație. Ca efect - următorul pas de algoritm care se execută este pasul k.

**EXERCITII**

1. "Merg la școală în fiecare zi înafară de sîmbătă și duminică".
Cum se poate formaliza această propoziție folosind operația de decizie alternativă ?
2. Aceeași problemă pentru propozițiile:
" Cîinele Azor este singurul meu prieten " ;
" Fratele meu poate citi și cărți fără poze " .

3. Se dă schema logică:



Să se scrie operația de decizie alternativă corespunzătoare.

4. Cele două operații de decizie alternativă de mai jos diferă doar prin poziția delimitatorilor. Ce diferențe apar la evaluare ?

i. dacă $x \geq 0$ atunci

dacă $x=0$ atunci $y \leftarrow 1$

altfel $y \leftarrow 0$

ii. dacă $x \geq 0$ atunci

dacă $x=0$ atunci $y \leftarrow 1$

altfel $y \leftarrow 0$

Să se traseze schemele logice corespunzătoare.

5. Se dau următoarele secvențe de operații pseudocod:

i. dacă p atunci B_1
 ■
 dacă nu p atunci B_2
 ■

ii. pp logic
 dacă p atunci pp --- nu p
 B_1
 ■
 dacă pp atunci B_2
 ■

Se poate identifica vreuna din ele cu

dacă p atunci B_1
 altfel B_2
 ■

De ce ?

*În condiții inițiale riguros determinate,
un algoritm face ce-i place !*

(Murphy)

III. CLASIFICAREA ALGORITMILOR

III.1 Algoritmi liniari

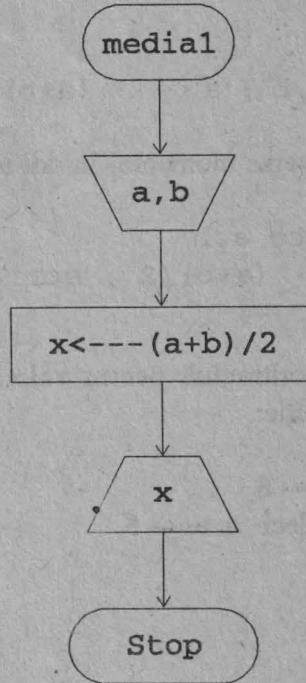
Aceștia constituie cea mai simplă clasă de algoritmi, neavînd nici un fel de operație de decizie. Execuția lor se face secvențial, într-o singură trecere; de aceea, de cele mai multe ori ei apar ca subalgoritmi în structuri mai complexe.

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. citește a, b
2.  $x \leftarrow (a+b)/2$ 
3. scrie x
4. Stop
```

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



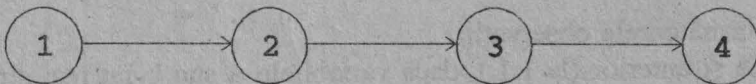
Dacă variabilele a și/sau b nu sînt utilizate în altă parte a algoritmului, atunci putem folosi pe oricare din ele în locul lui x:

- ```

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

```

Ordinea de parcurgere a pașilor este liniară:



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

1. citește  $a, b$  ;  $a \leftarrow (a+b)/2$  ; scrie  $a$  ; Stop

sau (spărgînd secvența la diverse momente) în doi sau trei pași, ca de exemplu

1. citește  $a, b$   
 2.  $a \leftarrow (a+b)/2$  ; scrie  $a$   
 3. Stop

Ca o aplicație a algoritmului, pentru valorile 2 și 8, parcurgerea lui conduce la următoarea execuție:

1.  $a \leftarrow 2$  ;  $b \leftarrow 8$   
 2.  $a \leftarrow (2+8)/2$  deci  $a \leftarrow 5$   
 3. scrie 5  
 4. Stop

## 1.2 Restul împărțirii a două numere întregi

rest  
 $r, a, b$  întregi /\*  $b$  nenul \*/  
 1. citește  $a, b$   
 2.  $r \leftarrow a - (a/b) * b$   
 3. scrie  $r$   
 4. Stop

Vom face următoarele observații:

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

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

Dacă nu sîntem atenți, o scriere neglijentă a pasului 2 poate genera erori deoarece în mulțimea numerelor întregi reprezentate în memoria unui calculator, **asociativitatea și distributivitatea nu mai funcționează.**

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

$$a - b * (a/b) , a - (a/b) * b , a - \text{int}(a/b) * b , \\ a - a/b * b , a - b * \text{int}(a/b)$$

iar greșite:

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

O execuție cu  $b \leftarrow 0$  duce la o blocare pe operația de împărțire (de tipul "număr prea mare"). De obicei se ia precauția 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  $a$  și  $b$  nu se cîtesc 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 operația **Stop** este înlocuită cu **Revenire**, pentru a asigura întoarcerea la algoritmul care apelează. În această situație,

**CITIREA ȘI SCRIEREA SÎNT CONTROLATE DE  
ALGORITMUL PRINCIPAL, NU DE SUBALGORITM.**

Să luăm ca exemplu valorile de intrare 15 și 9.

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

1.  $a \leftarrow 15$ ,  $b \leftarrow 9$

2.  $r \leftarrow 15/9*9 \equiv 15-1*9 \equiv 6$

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:

$a \leftarrow 15$ ,  $b \leftarrow 9$

1.  $r \leftarrow 15-15/9*9 \equiv 15-1*9 \equiv 6$

2. Revenire

și se revine în algoritm, în care variabila  $r$  are acum valoarea 6.

O condiție de forma  $a \geq b$  nu este necesară;

dacă  $a < b$  atunci  $a/b = 0$  și deci  $r \leftarrow a$ .

### 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 operația de atribuire. Dar, conform ei, după scrierea lui  $a \leftarrow b$ , valoarea care se afla anterior în  $a$  este ștearsă. Pentru a salva conținutul lui  $a$ , avem nevoie de o a treia variabilă  $c$  și atunci putem scrie:

perm\_1(a, b)

$a, b, c$  reale

1.  $c \leftarrow a$

2.  $a \leftarrow b$

3.  $b \leftarrow c$

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 inițială a lui  $a$  (aflată în  $c$ ) se depune în  $b$ .

Să luăm un exemplu: cum se execută  $\text{perm}_1(3, 0)$  :

|                               | $a$ | $b$ | $c$ |
|-------------------------------|-----|-----|-----|
| Inițial avem valorile         | 3   | 0   | -   |
| cu $c \leftarrow a$ se obține | 3   | 0   | 3   |
| cu $a \leftarrow b$ se obține | 0   | 0   | 3   |
| cu $b \leftarrow c$ se obține | 0   | 3   | 3   |

$c$  a fost folosită doar ca variabilă auxiliară și în final conținutul ei nu mai interesează.

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

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

Soluția poate fi de aceea numită și **regula celor trei pahare**.

De remarcat că efectul este același și în cazul secvenței

$$1. \quad c \leftarrow b ; b \leftarrow a ; a \leftarrow c$$

Folosind abilități de calcul, putem face permutarea celor două valori aflate în  $a$  și  $b$  **fără a fi nevoie de  $c$** . Pentru aceasta, să studiem secvența următoare:

- $\text{perm}_2(a, b)$
1.  $a \leftarrow a+b$
  2.  $b \leftarrow a-b$
  3.  $a \leftarrow a-b$
  4. Revenire

Inițial avem în locațiile lui a și b valorile  $\alpha$  respectiv  $\beta$ .

|                |           |                |                                      |
|----------------|-----------|----------------|--------------------------------------|
|                |           | a              | b                                    |
|                |           | $\alpha$       | $\beta$                              |
| După pasul     | 1. a ←--- | $a+b$          | $\beta$                              |
| Apoi           | 2. b ←--- | $\alpha+\beta$ | $(\alpha+\beta)-\beta \equiv \alpha$ |
| și, în sfârșit | 3. a ←--- | $a-b$          | $(\alpha+\beta)-\alpha \equiv \beta$ |

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

$$1. a \leftarrow 2+5 \equiv 7$$

$$2. b \leftarrow 7-5 \equiv 2$$

$$3. a \leftarrow 7-2 \equiv 5$$

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

Să presupunem că a și b sînt întregi (problema este aceeași pentru numere reale, doar că se lucrează cu valori mult mai mari). În calculator, numerele întregi nu trebuie să depășească 32 767 (altfel sînt necesare precauții speciale).

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

$$a+b \equiv 32767+3 \equiv 32770, \text{ număr care în a este reținut ca } -2.$$

$$\text{Deci, } a \leftarrow -2$$

$$\text{după care urmează } b \leftarrow -2-3 \equiv -5$$

$$a \leftarrow -2-(-5) \equiv 3,$$

numere total diferite de cele solicitate pentru a fi permutate.

**EXPLICAȚIE:** Oricărei variabile întregi i se asigură 16 poziții binare, din care prima codifică semnul (0 pentru numerele pozitive, 1 pentru cele negative).

Astfel, cel mai mare număr pozitiv care poate fi reținut este

$$0\ 111\ 1111\ 1111\ 1111 \text{ corespunzător scrierii în binar a lui } 32\ 767.$$

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

$$0\ 000\ 0000\ 0000\ 0011 \text{ și ajunge la } 1\ 000\ 0000\ 0000\ 0010,$$

corespunzător lui -2.



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

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

Se pot da doi algoritmi diferiți pentru rezolvarea acestei probleme:

```

valtr_1(a,b,c)
a,b,c,x,p reale
1. citește x
2. $p \leftarrow a \cdot x + b$
3. $p \leftarrow p \cdot x + c$
4. scrie p
5. Stop

```

```

valtr_2(a,b,c)
a,b,c,x,p reale
1. citește x
2. $p \leftarrow a \cdot x \cdot x + b \cdot x + c$
3. scrie x
4. Stop

```

În ambele variante am făcut distincție între coeficienții polinomului  $a, b, c$  (care, introduși dintr-un algoritm apelant dau contextului ideea că lucrăm cu constante) și variabila  $x$  (pentru care se calculează valoarea).

Să exemplificăm pentru trinomul  $p(x)=2x^2-x+3$  și  $x \leftarrow 1$ .

$\text{valtr}_1(2, -1, 3)$  va trece secvențial prin:

$a \leftarrow 2$ ,  $b \leftarrow -1$ ,  $c \leftarrow 3$

1.  $x \leftarrow 1$

2.  $p \leftarrow 2 \cdot 1 + (-1) \equiv 1$

3.  $p \leftarrow 1 \cdot 1 + 3 \equiv 4$

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

```

### Observații :

◆ Varianta `valtr_2` conține numai 4 pași față de `valtr_1`. Totuși, prima este mai interesantă, din două motive:

i. Dă un exemplu sugestiv al modului de utilizare a operației de asignare.

La pasul 4 se calculează

$$p*x+c \equiv (a*x+b)*x+c \equiv a*x*x+b*x+c$$

(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 înmulțiri și 2 adunări;  
varianta `valtr_2` - 3 înmulțiri și 2 adunări.

◆ În `valtr_1` pasul 2 se putea scrie și

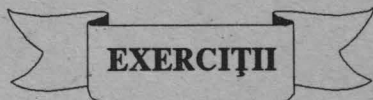
$$p \leftarrow a*x^2+b*x+c$$

În general însă este de preferat să se lucreze cu operația de înmulțire în loc de exponențiere deoarece se execută mai rapid și - în plus - la realizarea practică multe limbaje de programare nici nu posedă o astfel de operație.

◆ O scriere  $p(x)$  în loc de  $p$  este greșită deoarece, pentru ca  $p$  să fie o variabilă (conform operației de asignare) trebuie ca:

i.  $x$  să fie număr natural

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



## EXERCITII

1. Să se construiască algoritmi care să determine:
  - media geometrică a două numere;
  - media aritmetică a trei numere.
  
2. Se citește un număr  $a$ . Să se calculeze
  1.  $a^8$  folosind trei operații de înmulțire;
  2.  $a^6$  folosind trei operații de înmulțire;
  3.  $a^7$  folosind patru operații de înmulțire.
  
3. Se citește un număr real pozitiv  $a$ . Să se calculeze  $\sqrt[4]{a} \cdot \sqrt[8]{a^3}$ .  
 Se poate determina  $\sqrt[8]{a^5}$  folosind numai o înmulțire ?
  
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 și 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 obțină în aceleași variabile valorile  $(d, c, b, a)$ :
  - folosind o variabilă suplimentară de memorie. De cîte ori se folosește regula celor trei pahare ?
  - fără a folosi variabile suplimentare de memorie.
  
6. Să se dea exemple de cazuri cînd  
 $(a+b)+c \neq a+(b+c)$  ;  $a/b+c/b \neq (a+c)/b$  ;  $a*(b+c) \neq a*b+a*c$  ;

## 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  $\text{perm}_3(a,b)$  permută variabila  $a$  cu  $b$ ;
- ii. Atunci cînd  $\text{perm}_2(a,b)$  dă depășire,  $\text{perm}_3(a,b)$  va da rezultatul corect;
- iii. Cînd nu se poate aplica  $\text{perm}_3(a,b)$  ?

## 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 condiții asigură el permutarea conținutului celor două variabile ?

9. Se dau  $a$  și  $b$  întregi. Fără a folosi variabile suplimentare, să se dea un algoritm care să rețină în variabila  $a$  valoarea lui  $b$ , iar în  $b$  - valoarea restului împărțirii lui  $a$  la  $b$ .

10. Să se calculeze valoarea funcției

$$f(x) = \frac{a \cdot x + b}{c \cdot x + d} .$$

**III.2 ALGORITMI CU RAMIFICAȚII**

Această clasă de algoritmi se caracterizează prin două proprietăți:

- apar operațiile de decizie alternativă ca singure operații 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 toți pașii algoritmului.

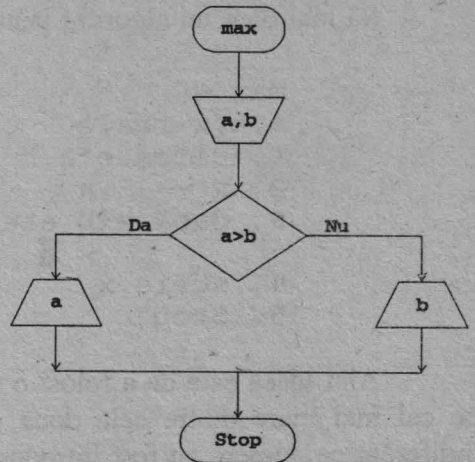
Dacă există o reprezentare sub formă de schemă logică, atunci vom avea doar un număr finit de drumuri de la primul la ultimul nod al schemei.

**2.1 Aflarea maximumului între două sau trei numere**

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

max\_1  
a, b reale

1. citește 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 greșită: în cazul când  $a$  este cel mai mare, după scrierea sa la pasul 2, se trece la pasul 3 unde se scrie și  $b$  (deci apar scrise ambele numere).

Pentru a corecta această greșeală, 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  $a \geq b$  s-ar scrie  $a$ , varianta la fel de corectă.

Să mai dăm un algoritm pentru aceeași problemă:

- max\_2  
 $a, b, x$  reale
1. citește  $a, b$
  2.  $x \leftarrow a$
  3. dacă  $x < b$  atunci  $x \leftarrow b$
  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  $\text{max\_1}$  prezintă o serie de avantaje în comparație cu  $\text{max\_2}$ :

|                   | $\text{max\_1}$ | $\text{max\_2}$ |
|-------------------|-----------------|-----------------|
| număr pași        | 3               | 5               |
| total variabile   | 2               | 3               |
| operații asignare | -               | 2               |
| operații decizie  | 1               | 1               |
| operații ieșire   | 2               | 1               |

Varianta  $\text{max\_2}$  se bucură însă de un atu pe care îl vom exploata ulterior: poate fi generalizată la un număr arbitrar de variabile.

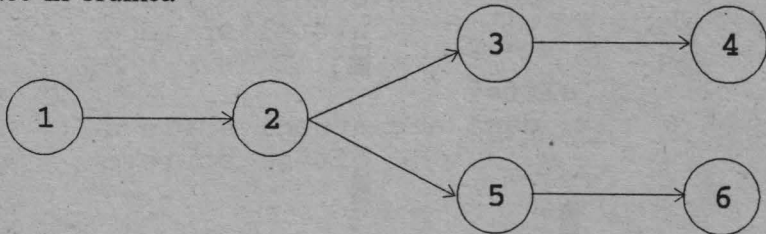
Pentru 3 numere, să folosim algoritmul  $\text{max\_1}$  în care vom adăuga și variabila c. Se obține:

```

max_3
a,b,c reale
1. citește 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

```

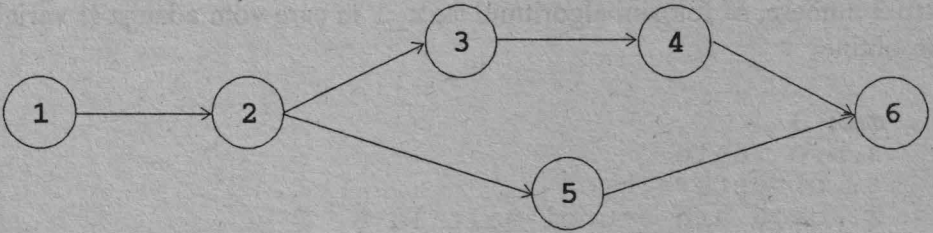
Deci, numărul de pași ai algoritmului s-a dublat. Trecerea de la unul la altul se face în ordinea



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

4'. salt la 6

ceea ce conduce la o singură operație **Stop** și la structura

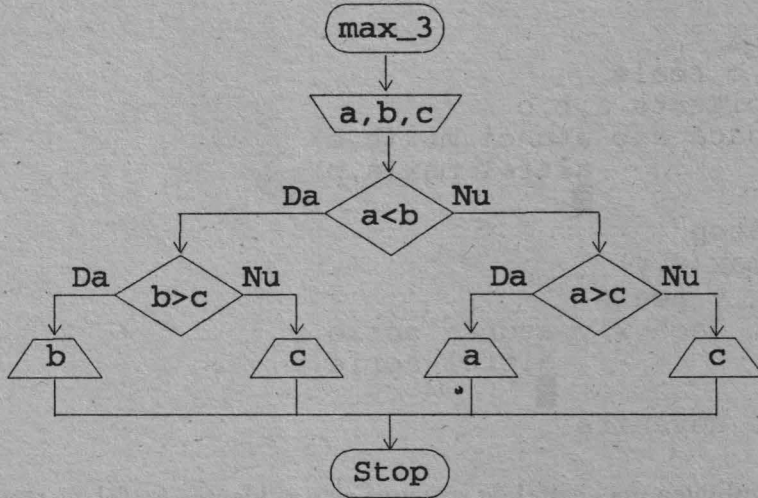


Numărul de pași se poate reduce printr-o scriere mai eficientă, în care vom elimina operațiile de salt fără condiție cum ar fi:

1. citește 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.  $a \leftarrow 2$  ,  $b \leftarrow 0$  ,  $c \leftarrow -5$

2. Testul  $a < b$  are valoarea logică fals (  $2 < 0 \equiv \text{fals}$  ). În acest caz, conform operației de decizie alternativă forma scurtă, se trece la pasul 3.

3. Testul  $a > c$  are valoarea logică adevărat (  $2 > -5 \equiv \text{adevărat}$  ), 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.  $a \leftarrow 0$  ,  $b \leftarrow 8$  ,  $c \leftarrow 1$

2.  $(a < b) \equiv (0 < 8) \equiv \text{adevărat} \Rightarrow$  se trece la pasul 5

5.  $(b > c) \equiv (8 > 1) \equiv \text{adevărat}$  și se scrie b, unde se află valoarea 8.

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

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

Obținem astfel:

```

max_4
a,b,c reale
1. citește 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 execuția algoritmului max cu asignările

$x \leftarrow b$  ,  $y \leftarrow c$ .

Altfel, subalgoritmul max este activat avînd valorile

$x \leftarrow a$  ,  $y \leftarrow b$  .

## 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. citește a,b
2. dacă $a \geq b$ atunci salt la 4
 █
3. $c \leftarrow a$; $a \leftarrow b$; $b \leftarrow c$
4. $c \leftarrow a - a/b * b$
5. dacă $c = 0$ atunci scrie a' este divizibil cu ',b
 altfel scrie a' nu este divizibil cu ',b
 █
6. Stop

```

De reținut:

- ◇ 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.

Același lucru îl obținem dacă scriem pasul 3 astfel:

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

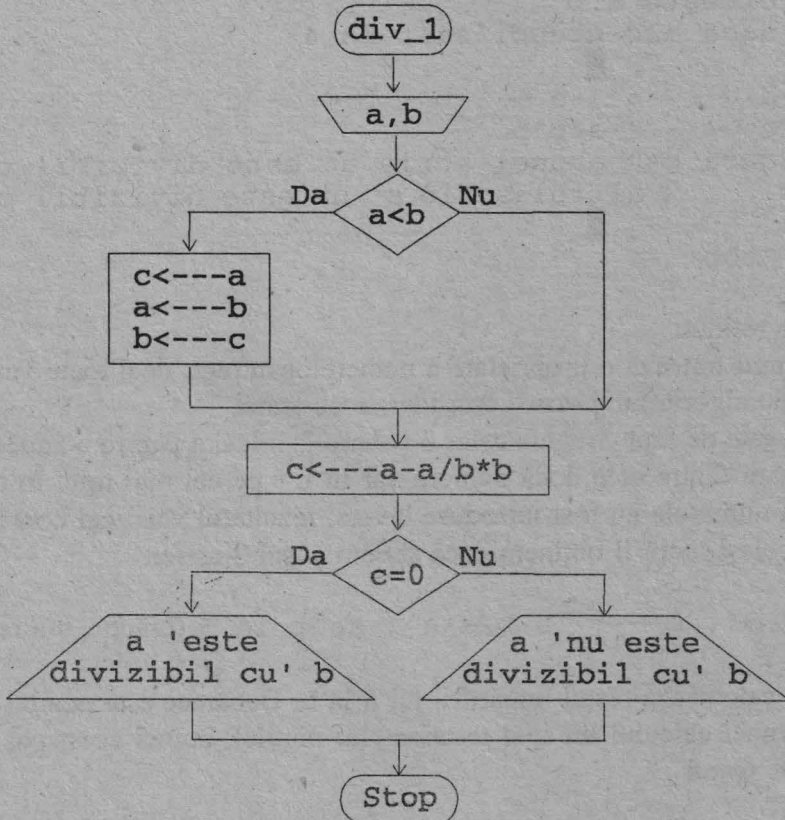
- ◇ Pasul 4 calculează restul împărțirii lui  $a$  la  $b$ . Deoarece este posibil să avem  $a=b$  (și atunci calculul lui  $c$  și testarea sînt inutile), atunci acest pas poate fi precedat de testul

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

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

❖ În cele două operații de scriere, în afara variabilelor de ieșire a fost inserat un text. Scris între semnele ' ', el va completa informația solicitată de utilizator.

Să prezentăm și schema logică corespunzătoare acestui algoritm :



◇ Verificări:

i.  $a \leftarrow 8$ ,  $b \leftarrow 3$ ; se trece prin:

2.  $(a \geq b) \equiv (8 \geq 3) \equiv \text{adev\c{a}rat} \Rightarrow 4$

4.  $c \leftarrow 8 - 8/3 * 3 \equiv 8 - 2 * 3 \equiv 2$

5.  $(c = 0) \equiv (2 = 0) \equiv \text{fals}$

**8 nu este divizibil cu 3.**

ii.  $a \leftarrow 5$ ,  $b \leftarrow 60$ :

2.  $(a \geq b) \equiv (5 \geq 60) \equiv \text{fals} \Rightarrow 3$

3. "regula celor trei pahare" conduce la  $a \leftarrow 60$ ,  $b \leftarrow 5$

4.  $c \leftarrow 60 - 60/5 * 5 \equiv 60 - 12 * 5 \equiv 0$

5.  $(c = 0) \equiv (0 = 0) \equiv \text{adev\c{a}rat}$

**60 este divizibil cu 5**

Algoritmul poate fi comasat în mai puşini paşii astfel:

div\_2

$a, b, c$  întregi

1. citeşte  $a, b$

2. dac   $a < b$  atunci  $c \leftarrow b$ ;  $b \leftarrow a$ ;  $a \leftarrow c$

3. dac   $a = a/b * b$  atunci scrie  $a'$  este divizibil cu  $b$   
 altfel scrie  $a'$  nu este divizibil cu  $b$

4. Stop

### 2.3 Rezolvarea ecuației de grad cel mult 2

Este exemplul tipic de algoritm cu ramificații, în special datorită condițiilor impuse de existența soluțiilor ecuației  $ax^2+bx+c=0$ .

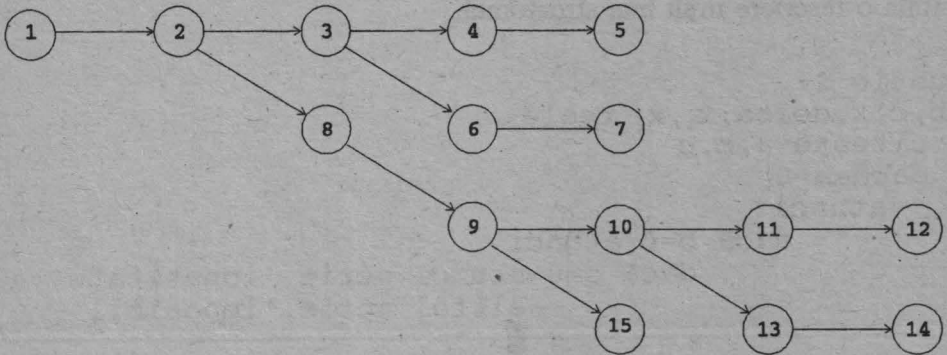
```

ecuație_1
a,b,c,x,delta,x1,x2 reale
1. citește 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 'ecuație imposibilă'
 █
5. Stop
6. x ←--- -c/b
7. scrie 'ecuație de gr.I cu soluția ',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 'soluție dublă ',x; Stop
15. scrie 'ecuația nu are rădăcini reale '; Stop

```

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

Structura algoritmului este :



Să cercetăm cum lucrează algoritmul pentru două seturi de date:

i. Ecuația  $x+2=0$ ; deci  $a \leftarrow 0$ ,  $b \leftarrow 1$ ,  $c \leftarrow 2$ ;

La pasul 2, condiția  $a \neq 0$  are valoarea logică fals, deci se continuă cu pasul 3. Aici  $b \neq 0$  are valoarea logică adevărat, ceea ce determină salt la pasul 6.

Se calculează  $x \leftarrow -2/1 \equiv -2$  după care se scrie:

ecuație de gradul I avînd soluția -2.

ii. Ecuația  $x^2-4x+2=0$ ; deci  $a \leftarrow 1$ ,  $b \leftarrow -4$ ,  $c \leftarrow 2$

2.  $(a \neq 0) \equiv (1 \neq 0) \equiv \text{adevărat} \Rightarrow 8$

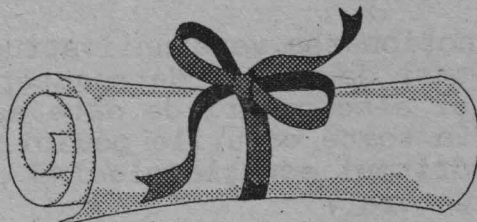
8.  $\text{delta} \leftarrow 16-4*1*2 \equiv 8$

9.  $(\text{delta} < 0) \equiv (8 < 0) \equiv \text{fals} \Rightarrow 10$

10.  $(\text{delta} = 0) \equiv (8 = 0) \equiv \text{fals} \Rightarrow 11$

11.  $x_1 \leftarrow (4-\text{rădăcina}(8))/2/1 \equiv 0.58578644$

$x_2 \leftarrow (4+\text{rădăcina}(8))/2/1 \equiv 3.4142136$



Pașii algoritmului ecuație\_1 pot fi rearanjați astfel încît să fie posibilă o rescriere mult mai structurată.

```

ecuație_2
a,b,c,x,delta,x1,x2 reale
1. citește 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 modalități de scriere și să se parcurgă următorul algoritm:

Test:

1. Încercați funcționarea versiunii structurate pentru cele două seturi de valori date anterior;
2. Introduceți și alte seturi de date astfel încît să se treacă prin toate cazurile posibile;
3. Refaceți algoritmul **ecuație** plecînd de la acest algoritm structurat;
4. Dacă nu ați reușit, salt la pasul 2 altfel Stop;



## 2.4 Algoritm cu ramificații pentru o problemă de căutare

Algoritmii matematici cu ramificații 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 mulțime finită A și doi interlocutori x și 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 mulțime A avînd  $2^k$  elemente.

Să particularizăm problema pentru lista:

- |    |            |            |              |              |
|----|------------|------------|--------------|--------------|
| A: | 1. Rață    | 5. Găină   | 9. Erete     | 13. Porumbel |
|    | 2. Colibri | 6. Condor  | 10. Pinguin  | 14. Struț    |
|    | 3. Ciine   | 7. Cal     | 11. Antilopă | 15. Leu      |
|    | 4. Rîmă    | 8. Fluture | 12. Șarpe    | 16. Greiere  |

joc\_1

x șîr de caractere

1. scrie 'Alegeți un nume de vietate din lista A. El poate fi ghicit din patru întrebări'; citește x
2. dacă x 'este pasăre' atunci salt la 3  
altfel salt la 10

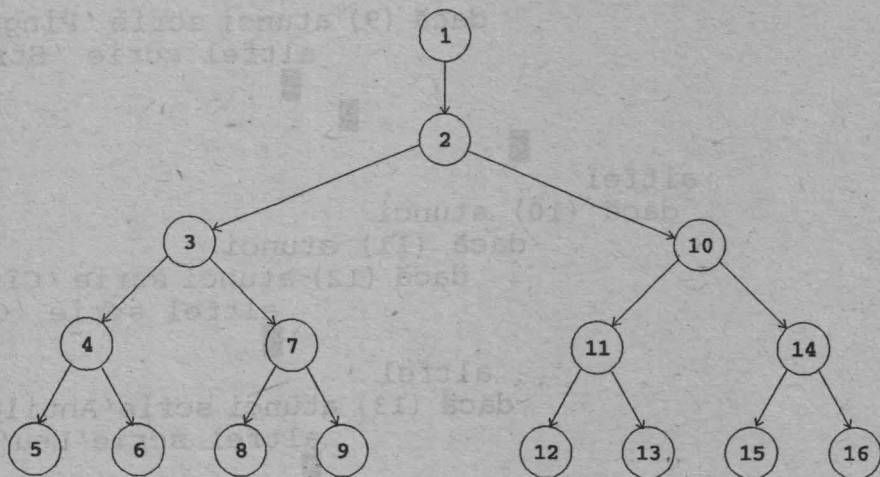


3. dacă x 'trăiește la noi în țară' atunci salt la 4  
altfel salt la 7
4. dacă x 'este domestică' atunci salt la 5  
altfel salt la 6
5. dacă x 'îi place apa' atunci scrie 'Rață'  
altfel scrie 'Găină'  
; Stop
6. dacă x 'este răpitor' atunci scrie 'Erete'  
altfel scrie 'Porumbel'  
; Stop
7. dacă 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 'trăiește la Pol' atunci scrie 'Pinguin'  
altfel scrie 'Struț'  
; 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 tîrăște' atunci salt la 15  
altfel salt la 16
15. dacă x 'mușcă' atunci scrie 'Șarpe'  
altfel scrie 'Rîmă'  
; Stop
16. dacă x 'cîntă' atunci scrie 'Greiere'

altfel scrie 'Fluture'  
 ; Stop

**Observații:**

- ◆ x nu este un număr ci un cuvânt (un șir de caractere); testele nu mai sînt exprimate prin expresii relaționale, 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 și construcția în care unele obiecte să fie găsite prin mai puțin (sau mai mult) de patru întrebări - în funcție de șansă. 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 rață?', x'este găină?', etc. Pentru situația 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 pașilor săi este intuitiv mai clară și constituie oarecum o justificare a termenului de "căutare binară":



Scris structurat, algoritmul se prezintă astfel:

```

joc_1
 1. citește x
 2. dacă (2) atunci
 dacă (3) atunci
 dacă (4) atunci
 dacă (5) atunci scrie 'Rață'
 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 'Struț'
 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'Șarpe'
 altfel scrie'Rîmă'
 altfel
 dacă (16) atunci scrie'Greiere'
 altfel scrie'Fluture'

```

### 3. Stop

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

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

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

Fie listele:

#### Lista 1

1. antilopă
2. erete
3. fluture
4. leu
5. pinguin
6. porumbel
7. șarpe
8. struț

#### Lista 2

1. cal
2. condor
3. găină
4. leu
5. porumbel
6. rîmă
7. șarpe
8. struț

#### Lista 3

1. antilopă
2. cal
3. ciine
4. colibri
5. condor
6. leu
7. pinguin
8. struț

#### Lista 4

1. antilopă
2. cal
3. ciine
4. erete
5. găină
6. leu
7. porumbel
8. rață

```

joc_2
x șir de caractere
i întreg
1. citește 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 poziția'i
 altfel scrie 'Greiere'
 ■
7. Stop

```

Ideea este bazată pe reprezentarea în binar a numerelor din intervalul  $[0, 2^4 - 1]$ . Fiecare test de apartenență 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 poziția numelui în lista A.

De exemplu, un nume care se află doar în listele 2 și 4 conduce la numărul binar 0101. Transformat în baza zece, acest număr este 5; pe poziția 5 în lista A se află "Găină". Într-adevăr, acest nume apare în listele 2 și 4 și 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 obține  $i \leftarrow 0$ ; poziția 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 semnificația lor.

Scrierea algoritmului joc2 se poate simplifica dacă folosim subalgoritmi:

joc\_3

1. citește x;  $i \leftarrow 0$

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  $i \leftarrow 16$

7. scrie 'Numele se află în lista A pe poziția' i

8. Stop

test (x,k,i)

1. dacă x este în lista k atunci  $i \leftarrow 2*i+1$   
 altfel  $i \leftarrow 2*i$

2. Revenire

## EXERCITII

1. Să se ordoneze crescător (descrescător) o secvență de 3 numere reale.
2. Se dau 4 numere întregi. Cîte din ele sînt impare ?

3. Să se rezolve sistemul de ecuații liniare  $a \cdot x + b \cdot y = c$ ,  $a' \cdot x + b' \cdot y = c'$ .
4. Se dau trei elemente ale unui triunghi (unghiuri, laturi). Să se determine celelalte trei elemente ale sale precum și 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 lângă 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 încât numărul de două cifre rămas să fie minim posibil.
8. Se dă un număr întreg. Câte cifre pare intră în componența lui ?
9. Se dă un interval  $[a, b]$  cu  $b - a = k$  și  $\alpha \in [a, b]$ . Să se determine un interval  $[x, y]$  cu proprietățile:  $\alpha \in [x, y]$ ,  $y - x = k/4$ .  
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 țări , iii) 32 nume de persoane.
11. Se dă un număr ales la întâmplare între 1 și 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 acești pași să fie parcurși 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. **Inițializarea (I)**: variabilele care intră în ciclu capătă valori cu care vor începe prima parcurgere.

ii. **Condiția de rămînere în ciclu (C)**: La fiecare parcurgere se verifică condiția inițială **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 condiției **C** trebuie făcută cu mare atenție; altfel este posibil ca ea să nu permită niciodată părăsirea ciclului și atunci nu mai avem un algoritm (nu mai este îndeplinită condiția de finitudine).

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

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

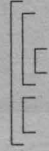
- **B** conține cel puțin un pas de algoritm;
- În timpul parcurgerii lui **B** trebuiesc modificate elementele care intră în condiția **C**; altfel **C** ia mereu aceeași valoare și bucla nu este părăsită.
- În **B** pot fi și alte cicluri.

Astfel, două bucle pot fi:

- a** distincte: nu au nici un pas comun:

[  
[

b) incluse una în alta :

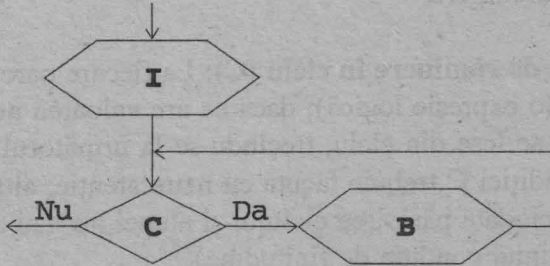


De remarcat că nu este permisă o utilizare a buclelor sub forma



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

1.

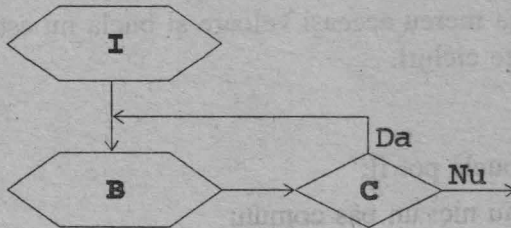


Această formă de schemă logică corespunde operației de algoritm

**I**  
 cât 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.

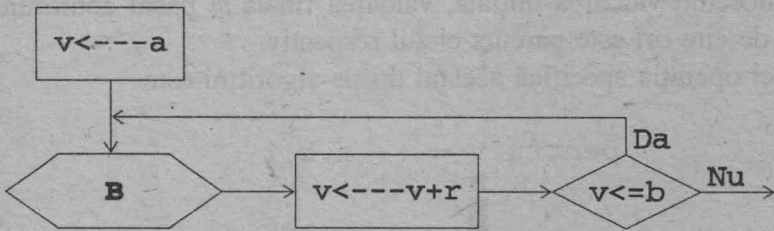


sau, tradus în operație de algoritm,

I  
repetă  
B  
pînă cînd (nu C)

Aici, corpul B este parcurs de algoritm cel puțin o dată.

Ca un caz particular al acestei variante, există posibilitatea



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

pentru  $v \leftarrow a, b, r$

B  
■

respectiv

pentru  $v \leftarrow a, b$

B  
■

dacă r are valoarea 1.

### 3.1 ALGORITMI CICLICI CU NUMĂR CUNOSCUȚ DE PAȘI

Sînt algoritmi ciclici în care numărul de treceri prin fiecare ciclu este fix. Mai sînt numiți și **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 inițială, valoarea finală și pasul contorului, putem determina de cîte ori este parcurs ciclul respectiv.

Deci operația 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

$$\max \{ 0, \text{int}((b-a)/r) \} + 1 \quad (*)$$

#### 3.1.1 Suma numerelor naturale pînă la $n$

```

suma_1
x, i, n întregi
1. citește 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  $n \leftarrow 4$ .

Primii doi pași asigură asignările  $n \leftarrow 4$  și  $x \leftarrow 0$ .

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

pas 3:  $i \leftarrow 0$ ,  $x \leftarrow x+i \equiv 0+0 \equiv 0$ .

După prima trecere prin buclă de la pasul 3 avem  $i \leftarrow 0$ ,  $x \leftarrow 0$ .

Urmează  $i \leftarrow i+1$ ; testul  $(i \leq n) \equiv (1 \leq 4) \equiv \text{adevărat}$  asigură reluarea pasului 3 cu  $i \leftarrow 1$ ,  $x \leftarrow 0$ . Avem

pas 3:  $x \leftarrow x+i \equiv 0+1 \equiv 1$

$i \leftarrow i+1 \equiv 1+1 \equiv 2$

$(i \leq n) \equiv (2 \leq 4) \equiv \text{adevărat}$  și pasul 3 se reia.

pas 3:  $x \leftarrow x+i \equiv 1+2 \equiv 3$

$i \leftarrow i+1 \equiv 2+1 \equiv 3$

$(i \leq n) \equiv (3 \leq 4) \equiv \text{adevărat}$

deci se revine la pasul 3 avînd asignările  $i \leftarrow 3$ ,  $x \leftarrow 3$ .

pas 3:  $x \leftarrow x+i \equiv 3+3 \equiv 6$

$i \leftarrow i+1 \equiv 3+1 \equiv 4$

$(i \leq n) \equiv (4 \leq 4) \equiv \text{adevărat}$

din nou, pasul 3 este reluat, cu  $i \leftarrow 4$  și  $x \leftarrow 6$ .

pas 3:  $x \leftarrow x+i \equiv 6+4 \equiv 10$

$i \leftarrow i+1 \equiv 4+1 \equiv 5$

$(i \leq n) \equiv (5 \leq 4) \equiv \text{fals}$

Condiția nu mai este adevărată și se iese din bucla de la pasul 3 cu

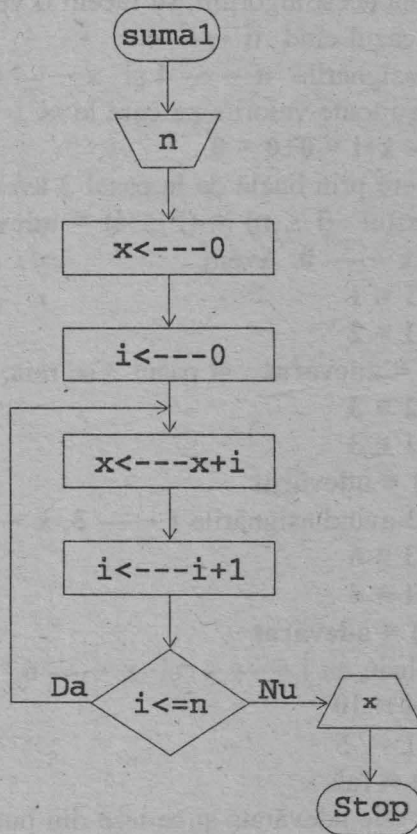
$x \leftarrow 10$ .

pas 4: scrie 10

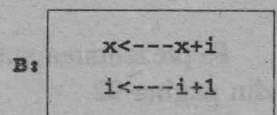
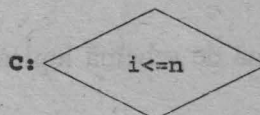
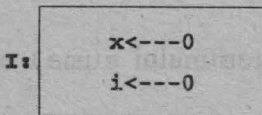
Stop

Într-adevăr,  $0+1+2+3+4=10$ .

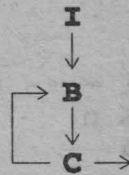
Reprezentarea sub formă de schemă logică a algoritmului suma1 este cea din pagina 92.



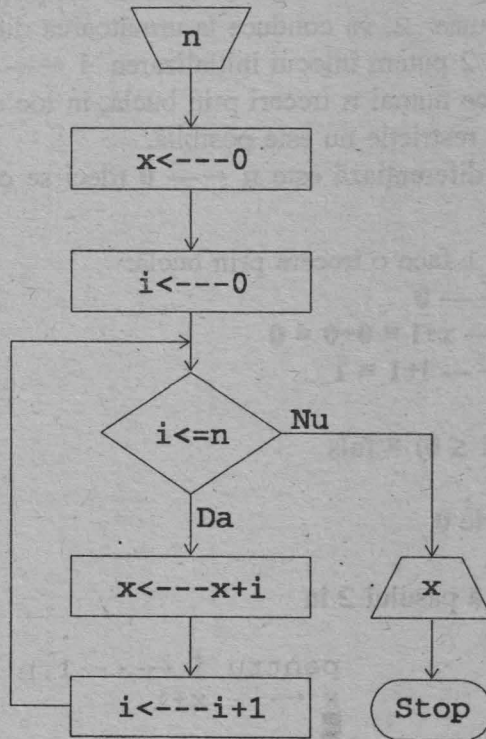
Cele trei blocuri caracteristice algoritmilor ciclici sînt:



și ele sînt așezate în ordinea



O ordonare inversă între blocurile B și C conduce la schema logică



Algoritmul scris pe baza acestei forme este :

```

suma_2
x,i,n întregi
1. citește n
2. $x \leftarrow 0$; $i \leftarrow 0$
3. cît timp $i \leq n$
 execută
 $x \leftarrow x+i$; $i \leftarrow i+1$
4. scrie x
5. Stop

```

Faptul că blocul **B** este executat totdeauna cel puțin odată, ceea ce nu este valabil pentru suma\_2, va conduce la următoarea diferențiere de scriere:

În suma\_2, la pasul 2 putem înlocui inițializarea  $i \leftarrow 0$  cu  $i \leftarrow 1$ .

Astfel, aici se vor face numai  $n$  treceri prin buclă, în loc de  $n+1$ .

La suma\_1, această restricție nu este posibilă.

Cazul limită care le diferențiază este  $n \leftarrow 0$  (deci se cere suma numerelor naturale pînă la 0).

Aici, suma\_1 face o trecere prin buclă:

```

 $i \leftarrow 0$
 $x \leftarrow x+i \equiv 0+0 \equiv 0$
 $i \leftarrow i+1 \equiv 1$

```

după care testarea

$(i \leq n) \equiv (1 \leq 0) \equiv \text{fals}$

va conduce la

scrie 0

O modificare a pasului 2 în

```

pentru $i \leftarrow 1, n$
 $x \leftarrow x+i$

```

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



$$x \leftarrow x+i \equiv 1$$

și rezultatul ar fi 1, fals.

Algoritmul suma\_2 în care modificăm pasul 2 în

$$2'. x \leftarrow 0 ; i \leftarrow 1$$

va începe pasul 3 cu testarea

$$(i \leq n) \equiv (1 \leq 0) \equiv \text{fals},$$

rezultat care încheie intrarea în buclă.

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

#### 4. scrie 0 .

Dacă  $n \geq 1$ , modificarea de la suma\_1 nu va afecta rezultatul. De aceea putem reface primul algoritm astfel:

```

suma_3
x, i, n întregi
1. citește 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. citește 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 ținem cont de operațiile caracteristice algoritmilor ciclici,

```

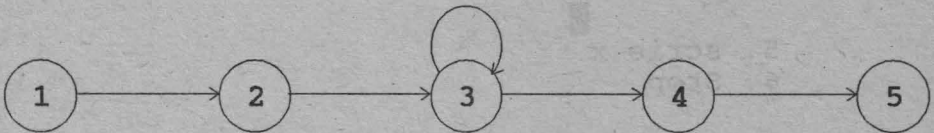
suma_5
x, i, n întregi
1. citește x
2. x ←--- 0 ; i ←--- 0
3. x ←--- x+i ; i ←--- i+1
4. dacă i ≤ n atunci salt la 3
 altfel scrie x
5. Stop

```

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

## O SCHEMĂ LOGICĂ REPREZINTĂ O CLASĂ DE ALGORITMI

Structura algoritmilor suma\_1, suma\_2 și suma\_4 este:



iar pentru suma\_5,



### Variabilele folosite:

◆ **Variabila de intrare**  $n$ ; dă numărul la care se oprește calculul sumei. După cum am văzut la diferențele între  $\text{suma}_1$  și  $\text{suma}_2$  modificată, o atenție deosebită trebuie acordată anumitor valori limită ale variabilei de intrare.

De asemenea este recomandabil ca algoritmul să conțină testări de verificare a corectitudinii datelor de intrare (de forma  $n \geq 0$ ).

◆ **Variabila contor**  $i$ ; în acești algoritmi ea are o valoare inițială  $i \leftarrow 0$  (sau  $i \leftarrow 1$  la  $\text{suma}_2$  modificată), o valoare finală  $i \leftarrow n+1$  care determină părăsirea buclei, și un pas 1.

De remarcat că o operație pentru include implicit în ea și asignarea  $i \leftarrow -1+1$ ; în cazul altor operații iterative (cît timp, repetă, dacă-atunci), această mărire a contorului trebuie scrisă efectiv.

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

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

```

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

```

Aici, deoarece pasul este  $-1$  (și nu  $1$ ), el a trebuit să fie scris explicit. În acest caz, valoarea inițială a contorului a fost  $n$  iar valoarea finală  $-1$ .

Pentru  $n \leftarrow 4$  se obține pe rînd

$$\begin{aligned}
 x &\leftarrow 4, x \leftarrow 4+3 \equiv 7, x \leftarrow 7+2 \equiv 9, \\
 x &\leftarrow 9+1 \equiv 10, x \leftarrow 10+0 \equiv 10.
 \end{aligned}$$

Și aici, dacă încercăm să optimizăm numărul de treceri prin buclă făcând inițializarea  $x \leftarrow n$  și plecând cu  $i \leftarrow n-1, 0, -1$ , algoritmul nu va funcționa în cazul particular  $n \leftarrow 0$ .

◆ **Variabila internă**  $x$  care este utilizată și ca **variabilă de ieșire**; constituie zona de bază pentru calcule. În ea se rețin toate sumele parțiale 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 inițializarea acestei variabile; ea este  $x \leftarrow 0$  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 inițial cu orice număr, îl lasă nemodificat (un astfel de număr se numește **element neutru**). Acesta nu poate fi decît 0.

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

```

suma_7
x, i, n întregi
1. citește 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 și recomandabilă.

### 3.1.2 Cite elemente dintr-un șir sînt mai mari decît o valoare dată

Avem de rezolvat următoarea problemă:

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

Problema este bineînțeleasă aceeași dacă cerem numărul elementelor "mai mici decît  $a$ " sau "egale cu  $a$ ". În marea majoritate a aplicațiilor practice,  $a \leftarrow 0$ .

Algoritmul prezentat va folosi o variabilă de lucru  $x$  în care vom citi pe rînd fiecare din cele  $n$  numere date.

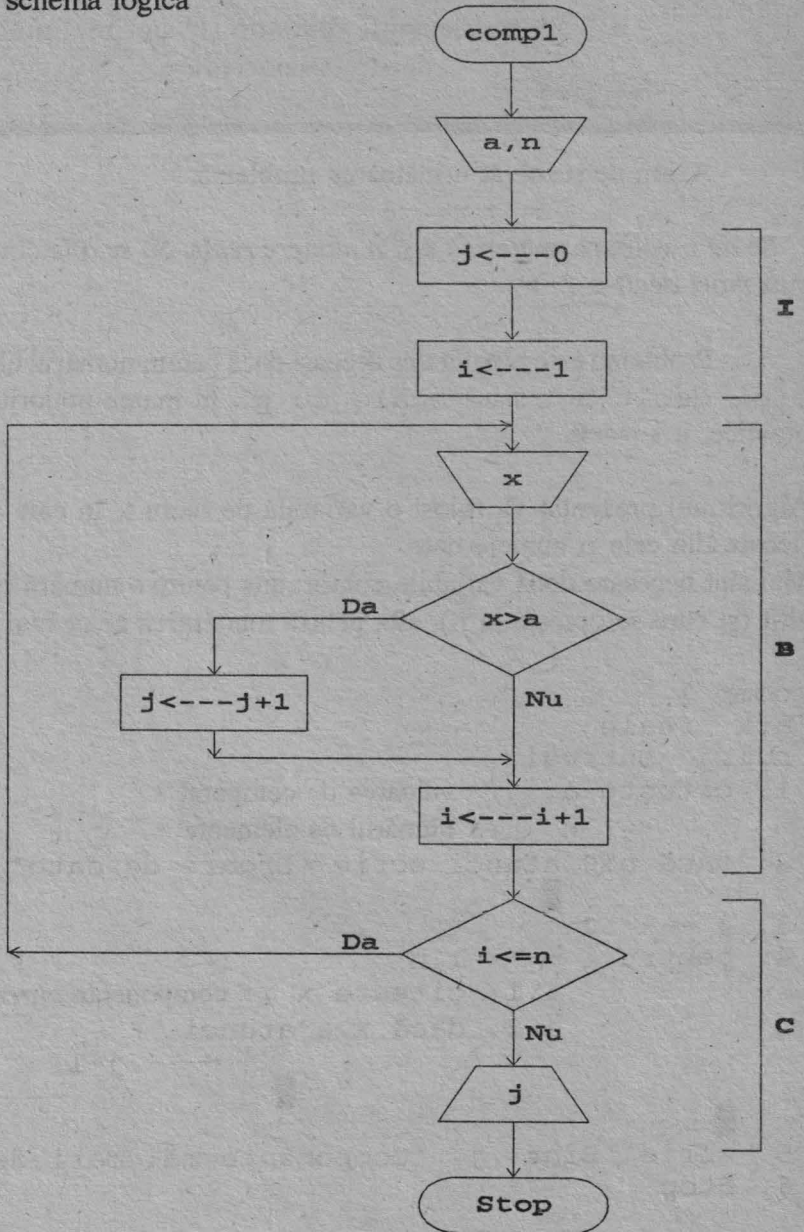
Mai sînt necesare două variabile-contor: una pentru a număra cîte variabile s-au citit (și care se oprește la  $n$ ), alta pentru numărarea celor mai mari decît  $a$ .

```

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

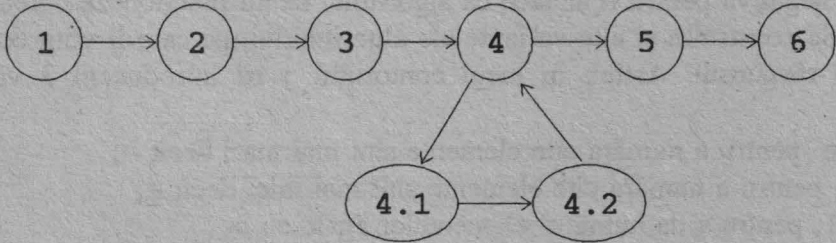
```

caracterizat prin schema logică



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

Structura algoritmului este:



Să luăm ca exemplu 3 numere ( $n \leftarrow 3$ ): 2, -5, 1 și  $a \leftarrow 0$ .

Deci problema este:

*Câte numere din cele trei sînt pozitive ?*

pas 1:  $a \leftarrow 0$ ,  $n \leftarrow 3$

pas 2:  $(n \leq 0) \equiv (3 \leq 0) \equiv \text{fals}$

pas 3:  $j \leftarrow 0$

pas 4:  $i \leftarrow 1$

4.1:  $x \leftarrow 2$  (se citește în  $x$  primul număr din cele 3);

4.2:  $(x > a) \equiv (2 > 0) \equiv \text{adevărat} \Rightarrow j \leftarrow j+1 \equiv 0+1 \equiv 1$

Ciclul s-a terminat, deci se revine la pasul 4:

pas 4:  $i \leftarrow i+1 \equiv 1+1 \equiv 2$

$(i \leq n) \equiv (2 \leq 3) \equiv \text{adevărat}$  și deci se poate reintra în ciclu;

4.1:  $x \leftarrow -5$  (se citește următorul număr);

4.2:  $(x > 0) \equiv (-5 > 0) \equiv \text{fals}$ ; din nou

pas 4:  $i \leftarrow i+1 \equiv 2+1 \equiv 3$

$(i \leq n) \equiv (3 \leq 3) \equiv \text{adevărat}$  și deci

4.1:  $x \leftarrow 1$  (se citește al treilea număr);

4.2:  $(x > a) \equiv (1 > 0) \equiv \text{adevărat} \Rightarrow j \leftarrow j+1 \equiv 1+1 \equiv 2$

În sfîrșit,

pas 4:  $i \leftarrow i+1 \equiv 3+1 \equiv 4$

$(i \leq n) \equiv (4 \leq 3) \equiv \text{fals}$  (nu mai sînt alte numere de citit),  
 și pasul 4 (bucla) este părăsit.

De remarcat că pasul 2 este doar o precauție asupra numărului  $n$ ; o valoare negativă pentru  $n$  ar face ca algoritmul să nu funcționeze corect.

Să construim și 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  $a$ .

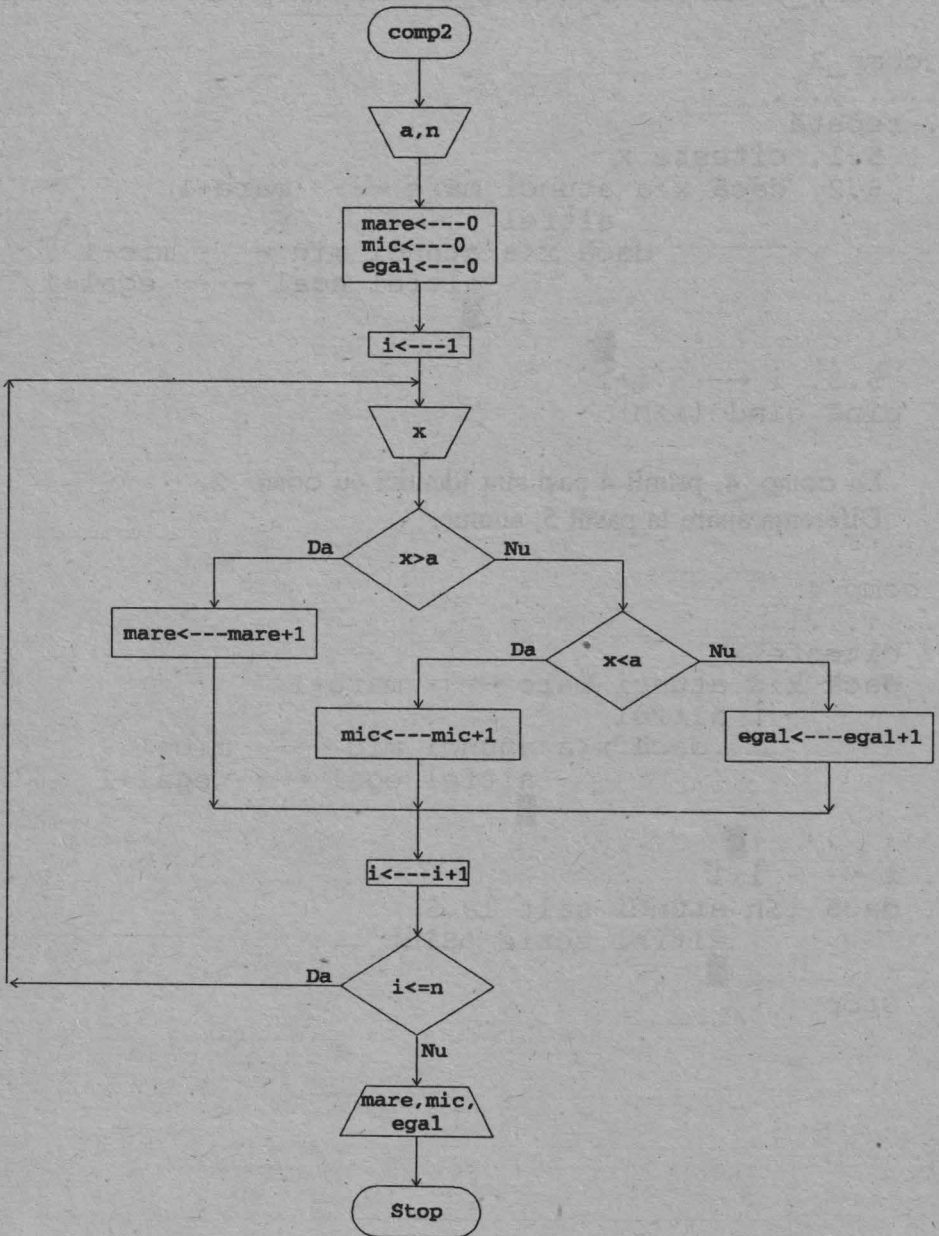
```

comp_2
a, x reale
n, i, mare, mic, egal întregi
1. citește a, n
2. dacă $n \leq 0$ atunci scrie 'Eroare de date'; Stop
3. mare ← 0 ; mic ← 0 ; egal ← 0
4. i ← 1
5. cît timp $i \leq n$
 execută
 5.1. citește 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 operația pentru alte operații posibile. Structura de bază este însă aceeași, anume schema logică următoare:





comp\_3 este identic cu comp\_2 înafară de pasul 5 care se scrie:

```

comp_3
.....
5. repetă
 5.1. citește 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 pași sînt identici cu comp\_2.  
Diferența apare la pasul 5, anume:

```

comp_4
.....
5. citește 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 enunțăm pentru început problema în termenii următori:

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

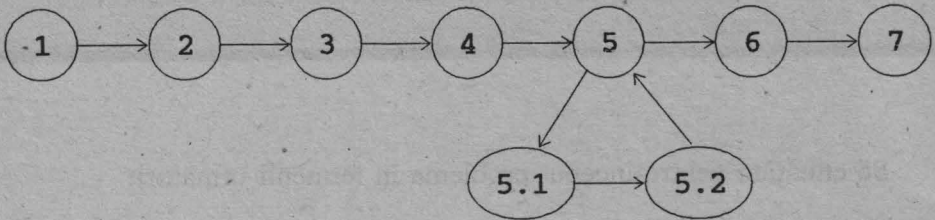
Ca și paragraful precedent, vom folosi două variabile de lucru: una (notată aici max) pentru a reține cel mai mare număr, și o alta - x, pentru a citi pe rînd cîte un număr din cele n.

```

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

```

Structura algoritmului este :



Lăsăm ca exercițiu construcția schemei logice corespunzătoare.

De exemplu, fie  $n \leftarrow 4$  și numerele 2, -3, 5, 0. Atunci:

pas 1:  $n \leftarrow 4$

pas 2:  $(n \leq 0) \equiv (4 \leq 0) \equiv \text{fals}$

pas 3:  $\max \leftarrow 2$  (primul număr este introdus în max)

pas 4:  $(n = 1) \equiv (4 = 1) \equiv \text{fals}$

pas 5:  $i \leftarrow 1$

5.1:  $x \leftarrow -3$  (se introduce al doilea număr în x)

5.2:  $(x > \max) \equiv (-3 > 2) \equiv \text{fals}$

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

pas 5:  $i \leftarrow i+1 \equiv 1+1 \equiv 2$

$(i \leq n-1) \equiv (2 \leq 3) \equiv \text{adevărat}$

5.1:  $x \leftarrow 5$

5.2:  $(x > \max) \equiv (5 > 2) \equiv \text{adevărat} \Rightarrow \max \leftarrow 5$

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

pas 5:  $i \leftarrow i+1 \equiv 2+1 \equiv 3$

$(i \leq n-1) \equiv (3 \leq 3) \equiv \text{adevărat}$

5.1:  $x \leftarrow 0$  (se introduce ultimul element).

5.2:  $(x > \max) \equiv (0 > 5) \equiv \text{fals}$

Și după această trecere prin buclă, max are valoarea 5.

pas 5:  $i \leftarrow i+1 \equiv 3+1 \equiv 4$

$(i \leq n-1) \equiv (4 \leq 3) \equiv \text{fals}$  și se iese din ciclul.

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 citește direct în max iar celelalte - pe rînd în x. Motivul este acela că la început este nevoie de o inițializare a variabilei max (și în mod evident, cel mai mare număr după citirea unuia singur, este acel număr). După ce se face această inițializare, 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 - și deci trebuie reținut în max.

Construcția nu funcționează în cazul limită  $n \leftarrow 1$ . Atunci, după citirea în max, algoritmul trebuie să treacă direct la scriere, deoarece nu mai sînt alte elemente de citit, numărul - unic - fiind evident și maxim.

Această observație impune introducerea pasului 4, care să elimine intrarea în ciclu pentru cazul  $n \leftarrow 1$ .

Totuși, pasul 4 din maxim1 poate fi evitat astfel:

3. citește max ;  $i \leftarrow 1$

4. cît timp  $i \leq n-1$

    execută

    4.1. citește x

    4.2. dacă  $x > \max$  atunci  $\max \leftarrow x$

    4.3.  $i \leftarrow i+1$

5. scrie 'Cel mai mare număr este ', max

6. Stop

Acum, dacă  $n \leftarrow 1$ , atunci intrarea în ciclul 4 se face cu inițializarea  $i \leftarrow 1$  și testul  $(i \leq n-1) \equiv (1 \leq 0) \equiv \text{fals}$ , deci se trece direct mai departe la pasul 5.

Aparent, în această situație devine inutil și testul de la pasul 2. Afirmatia este falsă, pentru că dacă  $n \leftarrow 0$ , atunci pasul 4 nu se execută; DAR, la pasul 3, ce se citește în max ? O variabilă care nu aparține algoritmului și care în final este declarată drept element maxim.

Putem construi algoritmul într-o formă mai omogenă, folosind ca inițializare pentru max o valoare care nu poate fi niciodată maximă; o notăm  $-\infty$  și 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. citește n
2. dacă $n \leq 0$ atunci scrie 'Eroare de date'; Stop
3. max $\leftarrow -\infty$
4. pentru i $\leftarrow 1, n$
 4.1. citește x
 4.2. dacă $x > \text{max}$ atunci max $\leftarrow x$
5. scrie 'Cel mai mare număr este', max
6. Stop

```

Acum precauția de la pasul 4 din maxim\_1 nu mai este necesară, toate elementele (chiar dacă  $n \leftarrow 1$ ) citindu-se în cadrul ciclului.

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

Vom folosi pentru aceasta variabilele:

max - pentru elementul maxim

min - pentru elementul minim

kmax - câte elemente maxime sînt printre cele n numere

kmin - câte elemente minime sînt printre cele n numere.

Celelalte variabile folosite au aceeași semnificație ca și la algoritmul anterior.

```

maxmin_1
x,max,min reale
n,i,kmax,kmin întregi
1. citește 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. citește 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 și  $n \leftarrow -5$ .

Ca rezultat se va obține

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

O variantă a sa folosind subalgoritmi este:

```

maxmin_2
x,max,min reale
n,i,kmax,kmin întregi
1. citește n
2. dacă $n \leq 0$ atunci scrie 'Eroare de date'; Stop
3. $\max \leftarrow -\infty$; $\min \leftarrow \infty$
4. $k_{\max} \leftarrow 0$; $k_{\min} \leftarrow 0$
5. pentru $i \leftarrow 1, n$
 5.1. citește 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 $p \cdot a > p \cdot b$ atunci $b \leftarrow a$; $k \leftarrow 1$
 altfel
 dacă $a = b$ atunci $k \leftarrow k + 1$
2. Revenire

```



Construcția s-a bazat pe observația că testul  $\min\langle x \rangle$  se poate scrie  $\neg \min\langle -x \rangle$  deci  $\neg \min$  operează ca un maxim pentru elementele de forma  $-x$ . În acest fel, maximul și minimul se pot determina cu un singur subalgoritm.

Să verificăm  $\text{maxmin}_2$  pentru numerele 2, -3, 2.

pas 1:  $n \leftarrow 3$

pas 2:  $(n \leq 0) \equiv (3 \leq 0) \equiv \text{fals}$

pas 3:  $\max \leftarrow -\infty$ ;  $\min \leftarrow \infty$

pas 4:  $k_{\max} \leftarrow 0$ ;  $k_{\min} \leftarrow 0$

pas 5:  $i \leftarrow 1$

5.1:  $x \leftarrow 2$

5.2:  $\text{control}(x, \max, k_{\max}, 1)$

Algoritmul control va lucra cu variabilele

$a \leftarrow x \equiv 2$ ,  $b \leftarrow \max \equiv -\infty$ ,  $k \leftarrow k_{\max} \equiv 0$ ,  $p \leftarrow 1$ .

P.1:  $(p \cdot a > p \cdot b) \equiv (2 > -\infty) \equiv \text{adev\~{a}rat}$

$b \leftarrow 2$ ,  $k \leftarrow 1$

Deci revenirea în algoritmul principal se face cu

$\max \leftarrow 2$ ,  $k_{\max} \leftarrow 1$ .

5.3:  $\text{control}(x, \min, k_{\min}, -1)$

Operarea cu  $a \leftarrow x \equiv 2$ ,  $b \leftarrow \min \equiv \infty$ ,  $k \leftarrow k_{\min} \equiv 0$ ,  $p \leftarrow -1$  în subalgoritm conduce la:

P.1:  $(p \cdot a > p \cdot b) \equiv (-2 > -\infty) \equiv \text{adev\~{a}rat}$

$b \leftarrow 2$ ,  $k \leftarrow 1$

Deci  $\min \leftarrow 2$  și  $k_{\min} \leftarrow 1$ .

pas 5:  $i \leftarrow i+1 \equiv 1+1 \equiv 2$

$(i \leq n) \equiv (2 \leq 3) \equiv \text{adev\~{a}rat}$

5.1:  $x \leftarrow -3$

5.2:  $\text{control}(x, \max, k_{\max}, 1)$  a cărui efectuare nu duce la nici o modificare a datelor.

5.3.  $\text{control}(x, \min, k_{\min}, -1)$

Se intră în subalgoritm cu  $a \leftarrow -3$ ,  $b \leftarrow 2$ ,  $k \leftarrow 1$ ,  $p \leftarrow -1$ . Atunci

P.1:  $(p * a > p * b) \equiv (3 > -2) \equiv \text{adev\c{a}rat}$

$b \leftarrow a \equiv 3$ ,  $k \leftarrow 1$

Revenire are ca efect

$\min \leftarrow -3$ ,  $k_{\min} \leftarrow 1$ .

pas 5:  $i \leftarrow i+1 \equiv 2+1 \equiv 3$

$(i \leq n) \equiv (3 \leq 3) \equiv \text{adev\c{a}rat}$

5.1:  $x \leftarrow 2$

5.2: control ( $x$ ,  $\max$ ,  $k_{\max}$ )

Deci subalgoritmul debutează cu inițializările

$a \leftarrow 2$ ,  $b \leftarrow \max \equiv 2$ ,  $k \leftarrow k_{\max} \equiv 1$ .

P.1:  $(a > b) \equiv (2 > 2) \equiv \text{fals}$

$(a = b) \equiv (2 = 2) \equiv \text{adev\c{a}rat} \Rightarrow k \leftarrow k+1 \equiv 2$

P.2: Revenire:  $\max \leftarrow b \equiv 2$ ,  $k_{\max} \leftarrow k \equiv 2$

5.3. control ( $x$ ,  $\min$ ,  $k_{\min}$ ,  $-1$ ) care nu modifică nimic.

pas 5:  $i \leftarrow i+1 \equiv 3+1 \equiv 4$

$(i \leq n) \equiv (4 \leq 3) \equiv \text{fals}$  și bucla 5 este abandonată.

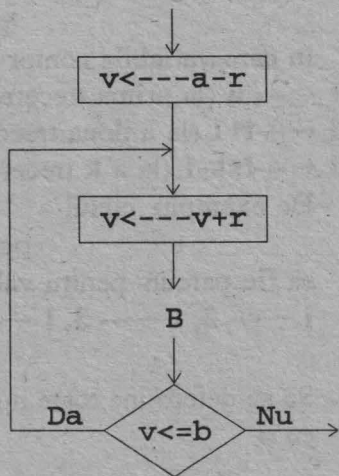
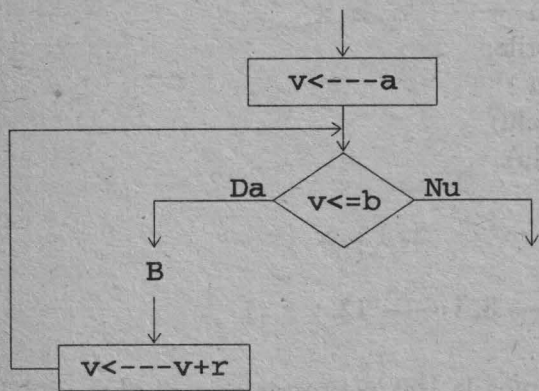
pas.5: scrie: Cel mai mare număr este 2. El apare de 2 ori.

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

## EXERCIIU

1. Să se scrie detaliat algoritmii `comp_3` și `comp_4` și 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 pași de algoritm bucele



4. Să se determine produsul numerelor naturale de la 1 la  $n$ .
5. Se dă un polinom  $p(x)$  și o valoare reală  $a$ . Să se calculeze  $p(a)$ .
6. Să se treacă un număr din baza  $b$  în baza 10.
7. Câți divizori pozitivi are un număr  $n$ ? Să se decidă apoi dacă  $n$  este prim.
8. Să se dea exemplu de construcție în care ciclul  
 pentru  $i \leftarrow a, b$   
 nu se termină.
9. Ce condiții verifică  $a$  și  $b$  pentru ca ciclul  
 pentru  $i \leftarrow a, b, -1$   
 $x \leftarrow x+a$   
 să nu se termine.

**10.** Să se imagineze un ciclu

pentru  $i \leftarrow \dots a, b, x$

în care variabila contor să ia valorile:

$i \leftarrow \dots a$  (la prima trecere prin ciclu)

$i \leftarrow \dots -i+1$  (la a doua trecere prin ciclu)

$i \leftarrow \dots -i+k-1$  (la a  $k$  trecere prin ciclu).

De exemplu, ciclul

pentru  $i \leftarrow \dots 2, 13, x$

să fie parcurs pentru valorile

$i \leftarrow \dots 2, i \leftarrow \dots 3, i \leftarrow \dots 5, i \leftarrow \dots 8, i \leftarrow \dots 12.$

**11.** Să se determine toate numerele  $a$  pînă la 100 cu proprietatea că  $a^2$  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.  $a^2+b^2=c^2$  (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 săptămîinii. Să se listeze toți anii din secolul XX care încep în acea zi a săptămîinii. 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 VARIABLE DE TIP TABLOU

Să ne gîndim la următoarea situație:

*În triajul unei gări se află vagoane; vagoane de călători, poștale, restaurant, de dormit, etc. Din toată această mulțime 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 și așezat 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 aceleași proprietăți (poartă același nume - numărul trenului, merg în aceeași direcție, se opresc în același loc, etc). Din acel moment proprietățile vagoanelor, deși se păstrează (sînt tot vagoane de clasă, poștale, de dormit, restaurant), se supun unei comenzi superioare, compunînd o nouă entitate - aceea de tren.

Aceeași situație o avem și în cazul variabilelor.

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

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

**ORICE GRUP DE VARIABILE CU PROPRIETĂȚI COMUNE ȘI CU UN SINGUR NUME SE NUMEȘTE TABLOU.**

Un tablou se caracterizează prin:

i. tipul variabilelor sale

**TOATE VARIABILELE UNUI TABLOU AU ACELAȘI 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 informație 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 același nume - a. Dintre ele, a [1] este prima variabilă, a [2] - a doua și a [3] - ultima.

În interiorul algoritmului, a [i] se folosește ca variabilă independentă (cu toate proprietățile corespunzătoare).

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

Pentru a facilita înțelegerea, 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 înțelegem 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.

**Atenție** (!) Să nu se facă confuzie între  $tab[3,4]$  care apare în zona de declarări și eventuala apariție a caracterului  $tab[3,4]$  în interiorul algoritmului. În definiție,  $tab[3,4]$  arată că există  $3 \cdot 4 = 12$  variabile reale avînd toate același nume  $tab$ , așezate 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).

**Observație:** 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 condiția ca într-un algoritm să nu existe simultan variabile simple și tablouri cu același nume.

Să prezentăm ca exemple cîteva 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:

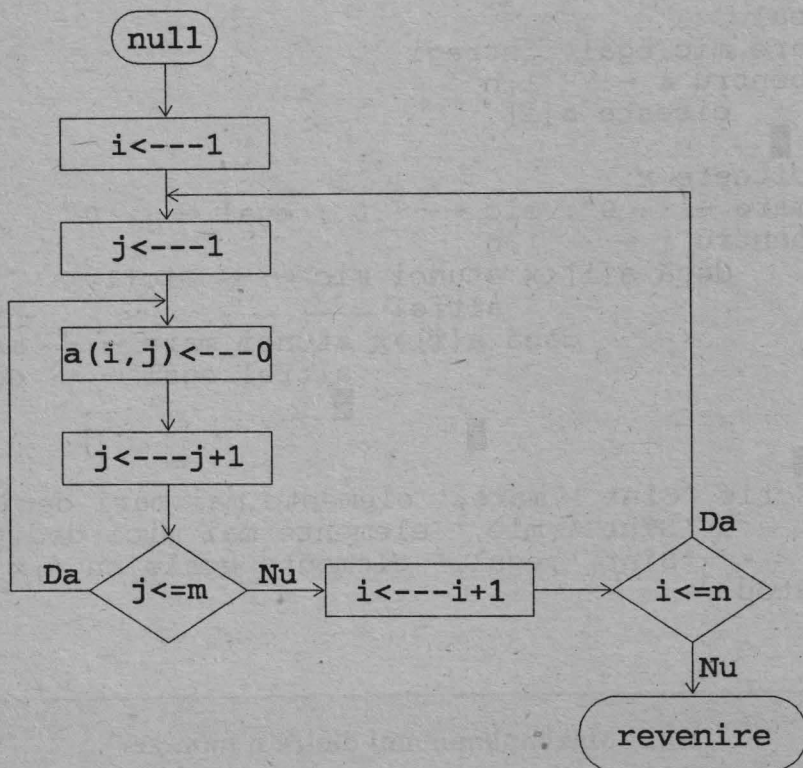
```

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 Cite elemente dintr-un șir sînt mai mari/mici decît x

Anterior am rezolvat această problemă fără a folosi tablouri, prin introducerea treptată a elementelor șirului; în acest fel însă, elementele se pierd și nu mai pot fi găsite pentru o altă eventuală utilizare. Vom da o soluție 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
 citește a[i]
 ■
2. citește 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

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

Facem mențiunea 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
 citește 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 și poziția 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
 citește a[i]
 ■
2. kmax ← 1; kmin ← 1;
 maxim[1] ← 1; minim[1] ← 1
3. max ← a[1]; min ← a[1]; i ← 2
4. cât 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 pozițiile : '

```

```

5.1. pentru i ←--- 1, kmax
 scrie maxim[i]

```

```

6. scrie ' Elementul minim este ', min
 ' .El se află în tablou pe pozițiile : '

```

```

6.1. pentru i ←--- 1, kmin
 scrie minim[i]

```

```

7. Stop

```

Elementele care trebuie analizate sînt în tabloul  $a$ .  $k_{\max}$  și  $k_{\min}$  dau numărul de elemente maxime respectiv minime. În tabloul maxim se introduc pozițiile pe care se află cele  $k_{\max}$  elemente maxime egale, iar în tabloul minim se introduc cele  $k_{\min}$  poziții ale elementelor minime egale.

Să exemplificăm algoritmul pentru tabloul  $a=(1.1, 12, -3, 12, 0)$ .

Aici  $n \leftarrow 5$ .

pas 1:  $a[1] \leftarrow 1.1, a[2] \leftarrow 12, a[3] \leftarrow -3, a[4] \leftarrow 12, a[5] \leftarrow 0$

pas 2:  $k_{\max} \leftarrow 1, k_{\min} \leftarrow 1, \text{maxim}[1] \leftarrow 1, \text{minim}[1] \leftarrow 1$

pas 3:  $\text{max} \leftarrow a[1] \equiv 1.1, \text{min} \leftarrow a[1] \equiv 1.1, i \leftarrow 2$

pas 4:  $(i \leq n) \equiv (2 \leq 5) \equiv \text{adev\~{a}rat}$

4.1:  $(a[i] > \text{max}) \equiv (12 > 1.1) \equiv \text{adev\~{a}rat}$

$kmax \leftarrow 1, maxim[1] \leftarrow 2, max \leftarrow 12$

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

4.2:  $(a[i] < min) \equiv (12 < 1.1) \equiv \text{fals}$

$(a[i] = min) \equiv (12 = 1.1) \equiv \text{fals}$

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

4.3:  $i \leftarrow i+1 \equiv 2+1 \equiv 3$

pas 4:  $(i \leq n) \equiv (3 \leq 5) \equiv \text{adev\~{a}rat}$

4.1:  $(a[i] > max) \equiv (-3 > 12) \equiv \text{fals}$

$(a[i] = max) \equiv (-3 = 12) \equiv \text{fals}$

4.2:  $(a[i] < min) \equiv (-3 < 1.1) \equiv \text{adev\~{a}rat}$

$min \leftarrow -3, kmin \leftarrow 1, minim[1] \leftarrow 3$

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

4.3:  $i \leftarrow i+1 \equiv 3+1 \equiv 4$

pas 4:  $(i \leq n) \equiv (4 \leq 5) \equiv \text{adev\~{a}rat}$

4.1:  $(a[i] > max) \equiv (12 > 12) \equiv \text{fals}$

$(a[i] = max) \equiv (12 = 12) \equiv \text{adev\~{a}rat}$

$kmax \leftarrow kmax+1 \equiv 1+1 \equiv 2, maxim[2] \leftarrow 4$

Printre primele 4 componente ale tabloului sînt două maxime situate pe pozițiile 2 și 4.

4.2:  $(a[i] < min) \equiv (12 < -3) \equiv \text{fals}$

$(a[i] = min) \equiv (12 = -3) \equiv \text{fals}$

4.3:  $i \leftarrow i+1 \equiv 4+1 \equiv 5$

pas 4:  $(i \leq n) \equiv (5 \leq 5) \equiv \text{adev\~{a}rat}$

4.1:  $(a[i] > max) \equiv (0 > 12) \equiv \text{fals}$

$(a[i] = max) \equiv (0 = 12) \equiv \text{fals}$

4.2:  $(a[i] < min) \equiv (0 < -3) \equiv \text{fals}$

$(a[i] = min) \equiv (0 = -3) \equiv \text{fals}$

4.3:  $i \leftarrow i+1 \equiv 5+1 \equiv 6$

pas 4:  $(i \leq n) \equiv (6 \leq 5) \equiv \text{fals}$

pas 5: Elementul maxim este 12.

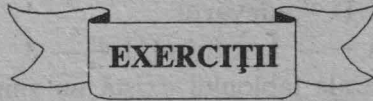
El se află în tablou pe pozițiile 2 și 4

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

Evident, algoritmul poate fi îmbunătățit.  
De exemplu, o facilitare ar putea fi făcută adăugînd la pasul 4.1., după  
 $\text{maxim}[1] \leftarrow \dots i$ , operația

salt la 4.3.

Lăsăm ca exercițiu justificarea acestei optimizări.



1. Să se rescrie algoritmi  $\text{maxim}_1$ ,  $\text{maxim}_2$  și  $\text{maxmin}_1$  folosind tablouri.
2. Se dă un număr  $n$ . Să se introducă cifrele sale într-un tablou.  
De exemplu, pentru numărul 31 se va obține  
 $\text{nr}[1] \leftarrow \dots 3$ ,  $\text{nr}[2] \leftarrow \dots 1$ .
3. Se dă un tablou cu  $n$  numere. Să se determine:
  - i. Cîte numere sînt pare;
  - ii. Cîte numere sînt într-un interval dat  $[a,b]$ .
4. Se dau  $n$  numere ( $n \geq 3$ ). Să se afle cele mai mari 3 numere dintre acestea.
5. Să se calculeze  $2^n$  pentru  $n$  între 31 și 50. Se știe că pentru aceste valori, implementarea algoritmului obișnuit (prin înmulțiri) 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 citește la fel de la stînga la dreapta cît și de la dreapta la stînga .  
Exemple: capac, sos, potop, epurașul ușa 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:  
 $b[1] \leftarrow a[1]$ ,  $b[2] \leftarrow a[1]+a[2]$ ,..., $b[n] \leftarrow a[1]+a[2]+..a[n]$ .
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 mulțimi;
  - Intersecția a două mulțimi;
  - Diferența a două mulțimi.
12. Se dă un tablou care conține o propoziție. Să se determine cîte cuvinte sînt în acea propoziție.
13. În aceeași ipoteză de mai sus, să se determine cîte cuvinte din propoziție încep cu o literă dată. Dar cîte cuvinte sînt formate din două litere ?

### 3.3 ALGORITMI CICLICI CU NUMĂR NECUNOSCUT DE PAȘI

În această categorie intră algoritmi 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 condiții.

Dintre operațiile 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 prezentați.

#### 3.3.1 Algoritmii 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 enunță astfel:

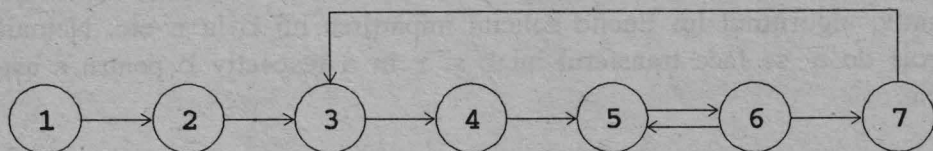
*Pentru două numere naturale  $a$  și  $b$  ( $a \geq b$ ) să se calculeze cel mai mare divizor comun (cmmdc) dintre  $a$  și  $b$ .*

```

euclid_1
a,b,r întregi
1. citește 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:

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

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

pas 1  $\Leftrightarrow$  pas 2

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

$$\mathbf{cmmdc}(a, b) = \mathbf{cmmdc}(b, a)$$

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  $\leftarrow \dots r$  operația

salt la 5

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

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

© Dacă pasul 3 lipsește din algoritm, acesta va continua să dea rezultate corecte !

Într-adevăr, dacă la pasul 1 se citesc  $a$  și  $b$  cu  $a < b$  și se ignoră pasul 3, se ajunge la pasul 5 unde obținem  $r \leftarrow a$  (deoarece  $a/b=0$ ). La pasul 6 se fac asignările  $a \leftarrow b$  și  $b \leftarrow r$ , deci se completează regula celor trei pahare obținându-se același efect ca la pasul 3.

© Variabila auxiliară utilizată în regula celor trei pahare de la pasul 3 este aceeași în pasul 5 (unde se calculează restul). Cei doi pași fiind distincți, nu apare nici o confuzie de notație.

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

Într-adevăr, se pare că la prima vedere, pașii 5 și 6 se pot restrânge în

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



Afirmația 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. citește a,b
```

2. dacă  $a \cdot b = 0$  atunci scrie 'Eroare de date'; Stop
3. cît timp  $b \neq 0$   
     execută  
      $a \leftarrow a + b - a/b \cdot b$ ;  $b \leftarrow a - b$ ;  $a \leftarrow a - b$
4. scrie a
5. Stop

Într-adevăr, să vedem ce rezultat va da euclid\_2 pentru  $a \leftarrow 12$ ,  $b \leftarrow 18$ . Repetăm:

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

pas 1:  $a \leftarrow 12$ ,  $b \leftarrow 18$

pas 2:  $(a \cdot b = 0) \equiv (12 \cdot 18 = 0) \equiv \text{fals}$

pas 3:  $(b \neq 0) \equiv (18 \neq 0) \equiv \text{adevărat}$

$$a \leftarrow a + b - a/b \cdot b \equiv 12 + 18 - 12/18 \cdot 18 \equiv 30$$

$$b \leftarrow a - b \equiv 30 - 18 \equiv 12$$

$$a \leftarrow a - b \equiv 30 - 12 \equiv 18$$

Deci, o permutare a lui a cu b fără nici o variabilă auxiliară.

Se revine la pasul 3 cu  $a \leftarrow 18$  și  $b \leftarrow 12$ .

pas 3:  $(b \neq 0) \equiv (12 \neq 0) \equiv \text{adevărat}$

$$a \leftarrow a + b - a/b \cdot b \equiv 12 + 18 - 18/12 \cdot 12 \equiv 30 - 12 \equiv 18$$

$$b \leftarrow a - b \equiv 18 - 12 \equiv 6$$

$$a \leftarrow a - b \equiv 18 - 6 \equiv 12$$

Noile valori sînt  $a \leftarrow 12$  și  $b \leftarrow 6$ .

pas 3:  $(b \neq 0) \equiv (6 \neq 0) \equiv \text{adevărat}$

$$a \leftarrow a + b - a/b \cdot b \equiv 12 + 6 - 12/6 \cdot 6 \equiv 18 - 12 \equiv 6$$

$$b \leftarrow a - b \equiv 6 - 6 \equiv 0$$

$$a \leftarrow a - b \equiv 6 - 0 \equiv 6$$

pas 3:  $(b \neq 0) \equiv (0 \neq 0) \equiv \text{fals}$

și se scrie a care are valoarea  $6 = \text{cmmdc}(12, 18)$ .

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

```

euclid_3
x,a,b întregi
1. citește 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 împărțiri, ținându-se cont de relația

$$\text{cmmdc}(a,b) = \text{cmmdc}(a-b,b)$$

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

$$\text{cmmdc}(a,a) = a$$

Cîteva observații:

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

Dacă dorim ca ea să apară pe ultima poziție 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 împărțire anunțând 'Număr prea mare'.

În varianta euclid\_3, pasul 3 se transformă în ciclu infinit, pentru că testul  $a > 0$  este totdeauna adevărat și se efectuează asignarea

$a \leftarrow a - 0.$

**c** Pașii 3 și 4 se pot scrie și

```

3. dacă $a > b$ atunci $a \leftarrow a - b$
 altfel
 dacă $a < b$ atunci $x \leftarrow a; a \leftarrow b; b \leftarrow x$
 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 și adăugăm

```

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

```

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

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

```

euclid_4
a, b întregi
1. citește 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 ceilalți algoritmi prezentați anterior.

### 3.3.3 Testarea dacă un număr este prim

Se știe că un număr  $n$  este prim dacă nu are alți divizori pozitivi înafară de 1 și  $n$ . Orice divizor diferit de  $+1, -1, +n, -n$  este un divizor propriu.

```

prim_1
i, n întregi
1. citește 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 deși există un contor  $i$ , numărul de treceri prin ciclul format de pasul 3 nu este cunoscut deoarece în momentul cînd se găsește un

divizor, acest ciclu este părăsit. Putem determina numai numărul maxim posibil de utilizări ale ciclului:  $\lfloor n/2 \rfloor + 1$ .

Este interesant că acest algoritm poate fi transformat într-unul cu număr cunoscut de pași. 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  $\max \{ \lfloor n/2 \rfloor - 1, 1 \}$  ori.

Dezavantajul acestui algoritm va fi acela că după ce găsim un divizor al lui  $n$ , deși 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. citește 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  $n \leftarrow 9000$ , deși de la prima trecere (la împărțirea 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 eficiență a algoritmului, o variantă care să decidă răspunsul mai rapid se poate construi plecînd de la următoarea proprietate (a cărei demonstrație o propunem ca exercițiu ):

(\*) Dacă  $n$  admite un divizor propriu pozitiv,  
atunci admite un divizor propriu  $\leq$  rădăcina( $n$ ).

```

prim_3
n,i,p întregi
1. citește n
2. dacă $n \leq 1$ atunci scrie 'Eroare de date'; salt la 1
3. dacă $n \leq 3$ atunci salt la 7
4. dacă $n = n/2 * 2$ atunci salt la 8
5. $p \leftarrow$ rădăcina(n) ; $i \leftarrow 3$
6. cît timp $i \leq p$
 execută
 dacă $n = n/i * i$ atunci salt la 8
 altfel $i \leftarrow i + 2$
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 pașii lui o formează testele de început; astfel

♦ pasul 2 asigură numere naturale nenule la intrare; în caz contrar se anunță eroare și se așteaptă asignarea altei valori pentru  $n$ .

De remarcat că se formează o buclă pas 1  $\Leftrightarrow$  pas 2

din care se iese numai dacă  $n$  primește o valoare strict mai mare ca 1.

♦ condiția  $n \leq 3$  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 împărțiri cu numere pare, dacă  $n$  nu este divizibil cu 2, celelalte verificări sînt inutile.

◆ De fapt pașii 2-4 se pot scrie într-unul singur astfel:

```

2' .dacă $n \leq 1$ atunci scrie 'Eroare de date'; salt la 1
 altfel
 dacă $n \leq 3$ 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  $[3, \text{radacina}(n)]$ .

Numărul exact de treceri prin buclă nu este fix, dar limitat la maximum  $[\text{radacina}(n)] - [\text{radacina}(n/2)] - 1$  (recomandăm o demonstrație teoretică a acestui rezultat).

c. Deși rădăcina  $(n)$  nu este în general un număr întreg, prin operația de asignare

$p \leftarrow \text{rădăcina}(n)$ ,

în  $p$  se reține numai partea întreagă a rădăcinii.

d. Verificare pentru  $n \leftarrow 79$ . Avem

pas 5:  $p \leftarrow \text{rădăcina}(79) \equiv 8$ ;  $i \leftarrow 3$

pas 6:  $(i \leq p) \equiv (3 \leq 8) \equiv \text{adevărat}$   
 $(n = i * i) \equiv (79 = 78) \equiv \text{fals}$

pas 6 (altfel):  $i \leftarrow i + 2 \equiv 5$

pas 6:  $(i \leq p) \equiv (5 \leq 8) \equiv \text{adevărat}$   
 $(n = i * i) \equiv (79 = 75) \equiv \text{fals}$

pas 6 (altfel):  $i \leftarrow i + 2 \equiv 7$

pas 6:  $(i \leq p) \equiv (7 \leq 8) \equiv \text{adevărat}$

$(n=n/i*i) \equiv (79=77) \equiv \text{fals}$

pas 6 (altfel):  $i \leftarrow i+2 \equiv 9$

pas 6:  $(i \leq p) \equiv (9 \leq 8) \equiv \text{fals}$

și ciclul se încheie: **79 nu este număr prim .**

Algoritmii  $\text{prim}_1$  și  $\text{prim}_2$  conduceau la același rezultat, dar după 38 treceri prin buclă (față de numai 4 în acest caz).

Cînd se cer toate numerele prime pînă la  $n$ , algoritmul poate fi reformulat și mai eficient:

$\text{prim}_4$

$m, n, k, i, j$  întregi

1. citește  $n$

2. dacă  $n \leq 1$  atunci scrie 'Eroare de date'; salt la 1

3.  $m \leftarrow \text{rădăcina}(n)$

tablou  $\text{tab}[m]$  de întregi

4.  $\text{tab}[1] \leftarrow 2$ ;  $k \leftarrow 1$ ;  $i \leftarrow 1$ ;  $j \leftarrow 3$

5. dacă  $n=2$  atunci salt la 7

6. cît timp  $j \leq n$

    execută

        6.1. cît timp  $\text{tab}[i] \leq \text{rădăcina}(j)$

            execută

                dacă  $j = j / \text{tab}[i] * \text{tab}[i]$

                    atunci salt la 6.3

                    altfel  $i \leftarrow i+1$

        6.2.  $k \leftarrow k+1$ ;  $\text{tab}[k] \leftarrow j$

        6.3.  $i \leftarrow 2$ ;  $j \leftarrow j+2$

7. pentru  $i \leftarrow 1, k$

    scrie  $\text{tab}[i]$

8. Stop

O primă observație 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 împărțindu-l la toate numerele prime de la 3 la rădăcina ( $j$ ). În caz că nici o împărțire nu se face exact,  $j$  este prim și va fi trecut în `tab`. Altfel, cînd se găsește un divizor prim al lui  $j$ , această valoare a lui  $j$  este părăsită. Prima poziție 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 mulți algoritmi. Aceasta și deoarece domeniul de aplicabilitate este deosebit de vast. Algoritmii construiți sînt în general de tip mixt, utilizînd cicluri cu număr cunoscut sau necunoscut de pași.

Vom prezenta trei algoritmi diferiți, fără a avea intenția de a epuiza posibilitățile relative la această temă.

În cele ce urmează vom presupune că un tablou  $a=(a_1, a_2, \dots, a_n)$  este ordonat crescător dacă  $a_1 \leq a_2 \leq \dots \leq a_n$ .

```
ord_1
i, n, s întregi
x real
1. citește n
 tablou a[n] de reale
2. pentru i ←--- 1, n
 citește 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.
 ■
```

```

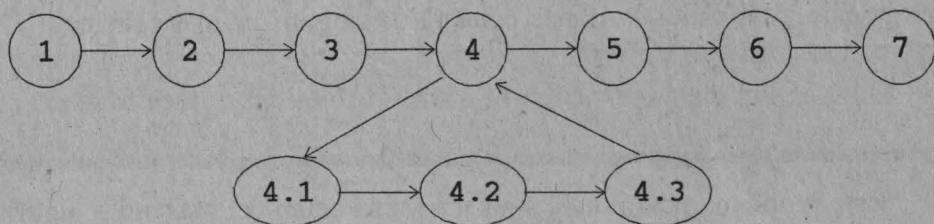
4.2. s ←--- 1; x ←--- a[i]; a[i] ←--- a[i+1];
 a[i+1] ←--- x
4.3. i ←--- i+1

```

5. dacă  $s=1$  atunci salt la 3

6 pentru  $i \leftarrow--- 1, n$   
scrie  $a[i]$

7. Stop



Comentarii: **a** Algoritmul `ord_1` conține două bucle, înafara celor de citire/scriere de la pașii 2 respectiv 6 :

■ cea de la pașul 4, în care contorul de ciclu este  $i$ . Bucla este parcursă de  $n-1$  ori (în cazul limită  $n \leftarrow--- 0$ , de zero ori).

■ pașii 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 puțin 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  $s \leftarrow--- 0$ ).

Deci avem de-a face cu un algoritm mixt.

**b** Să parcurgem algoritmul pentru tabloul  $a=(3, 5, 2, 3)$ ,

deci  $n \leftarrow--- 4$ .

pas 3:  $i \leftarrow--- 1; s \leftarrow--- 0$

pas 4:  $(i < n) \equiv (1 < 4) \equiv \text{adev\~{a}rat}$

4.1:  $(a[1] \leq a[2]) \equiv (3 \leq 5) \equiv \text{adev\~{a}rat}$

4.3:  $i \leftarrow i+1 \equiv 2$

pas 4:  $(i < n) \equiv (2 < 4) \equiv \text{adev\^ar\at}$

4.1:  $(a[2] \leq a[3]) \equiv (5 \leq 2) \equiv \text{fals}$

4.2:  $s \leftarrow 1, a[2] \leftarrow 2, a[3] \leftarrow 5$  (permutare)

4.3:  $i \leftarrow i+1 \equiv 3$

Deci acum tabloul este de forma (3, 2, 5, 3).

pas 4:  $(i < n) \equiv (3 < 4) \equiv \text{adev\^ar\at}$

4.1:  $(a[3] \leq a[4]) \equiv (5 \leq 3) \equiv \text{fals}$

4.2:  $s \leftarrow 1, a[3] \leftarrow 3, a[4] \leftarrow 5$  (permutare)

4.3:  $i \leftarrow i+1 \equiv 4$

și se obține  $a = (3, 2, 3, 5)$ .

pas 4:  $(i < n) \equiv (4 < 4) \equiv \text{fals}$ : închiderea buclei 4.

pas 5:  $(s=1) \equiv (1=1) \equiv \text{adev\^ar\at}$

pas 3:  $i \leftarrow 1, s \leftarrow 0$  și se reia ciclul pas 4

pas 4:  $(i < n) \equiv (1 < 4) \equiv \text{adev\^ar\at}$

4.1:  $(a[1] \leq a[2]) \equiv (3 \leq 2) \equiv \text{fals}$

4.2:  $s \leftarrow 1, a[1] \leftarrow 2, a[2] \leftarrow 3$  (permutare)

4.3:  $i \leftarrow i+1 \equiv 2$

Acum  $a = (2, 3, 3, 5)$ .

Celelalte parcurgeri prin cele două cicluri nu mai modifică tabloul.

Schimbarea condiției de la 4.1. în  $a[i] \geq a[i+1]$  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 obținem va fi însă un algoritm cu număr cunoscut de pași

```
ord 2
n, i, j întregi
x real
1. citește n
 tablou a[n] de reale
2. pentru i ← 1, n
 citește 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 aceleași date cu care am verificat algoritmul anterior,  
 $a = (3, 5, 2, 3)$  avem:

pas 3:  $j \leftarrow 1$

pas 4:  $(j < n) \equiv (1 < 4) \equiv \text{adev\c{a}rat}$

4.1:  $i \leftarrow 1$

4.2:  $(i < n+1-j) \equiv (1 < 4) \equiv \text{adev\c{a}rat}$

4.2.1:  $(a[i] > a[n+1-j]) \equiv (a[1] > a[4]) \equiv (3 > 3) \equiv \text{fals}$

4.2.2:  $i \leftarrow i+1 \equiv 2$

4.2:  $(i < n+1-j) \equiv (2 < 4) \equiv \text{adev\c{a}rat}$

4.2.1:  $(a[i] > a[n+1-j]) \equiv (a[2] > a[4]) \equiv (5 > 3) \equiv \text{adev\c{a}rat}$

$a[2] \leftarrow 3, a[4] \leftarrow 5$  (permutare)

Noul tablou este  $(3, 3, 2, 5)$ .

4.2.2:  $i \leftarrow i+1 \equiv 3$

4.2:  $(i < n+1-j) \equiv (3 < 4) \equiv \text{adev\c{a}rat}$

4.2.1:  $(a[3] > a[4]) \equiv (2 > 5) \equiv \text{fals}$

4.2.2:  $i \leftarrow i+1 \equiv 4$

4.2:  $(i < n+1-j) \equiv (4 < 4) \equiv \text{fals}$

4.3:  $j \leftarrow j+1 \equiv 2$

pas 4:  $(j < n) \equiv (2 < 4) \equiv \text{adev\text{a}rat}$

4.1:  $i \leftarrow 1$

4.2:  $(i < n+1-j) \equiv (1 < 3) \equiv \text{adev\text{a}rat}$

4.2.1:  $(a[i] > a[n+1-j]) \equiv (a[1] > a[3]) \equiv (3 > 2) \equiv \text{adev\text{a}rat}$

$a[1] \leftarrow 2, a[3] \leftarrow 3$  (permutare)

și se ajunge la (2, 3, 3, 5).

4.2.2:  $i \leftarrow i+1 \equiv 2$

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 găsește elementul maxim pentru tabloul de lungime  $n$  și-l depune în ultima componentă. Reia același procedeu pentru tabloul format cu primele  $n-1$  componente, și așa mai departe. Rămîne în final un tablou cu o componentă, al cărui element este evident maxim.

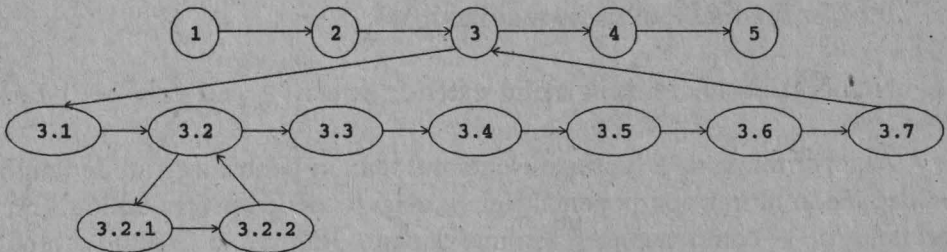
În sfîrșit, un al treilea algoritm pentru ordonarea crescătoare a elementelor tabloului este:

```
ord_3
n, i, j, p întregi
x real
1. citește n
 tablou a[n] de reale
2. i ← 2; citește a[1]
3. cît timp i ≤ n
 execută
 3.1. citește 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.  $p \leftarrow i+1$   
 3.4.  $p \leftarrow p-1$ ;  $a[p] \leftarrow a[p-1]$   
 3.5. dacă  $p > j+1$  atunci salt la 3.4.  
 3.6.  $a[j] \leftarrow x$   
 3.7.  $i \leftarrow i+1$

4. pentru  $i \leftarrow 1, n$   
 scrie  $a[i]$

5. Stop



Fie  $a=(3, 5, 2, 3)$ .

pas 2:  $i \leftarrow 2$ ,  $a[1] \leftarrow 3$  (citire)

pas 3:  $(i \leq n) \equiv (2 \leq 4) \equiv \text{adev\~{a}rat}$

3.1:  $x \leftarrow 5$  (citire),  $j \leftarrow 1$

3.2:  $(j < i) \equiv (1 < 2) \equiv \text{adev\~{a}rat}$

3.2.1:  $(x < a[j]) \equiv (5 < 3) \equiv \text{fals}$

3.2.2:  $j \leftarrow j+1 \equiv 2$

3.2:  $(j < i) \equiv (2 < 2) \equiv \text{fals}$

și bucla 3.2. se încheie.

3.3:  $p \leftarrow i+1 \equiv 3$

3.4:  $p \leftarrow p-1 \equiv 2$ ,  $a[2] \leftarrow a[1] \equiv 3$

3.5:  $(p > j+1) \equiv (2 > 3) \equiv \text{fals}$

3.6:  $a[2] \leftarrow 5$

3.7:  $i \leftarrow i+1 \equiv 3$

pas 3:  $(i \leq n) \equiv (3 \leq 4) \equiv \text{adev\~{a}rat}$

3.1:  $x \leftarrow 2$  (citire),  $j \leftarrow 1$

3.2:  $(j < i) \equiv (1 < 3) \equiv \text{adev\~{a}rat}$



- 3.2.1:  $(x < a[j]) \equiv (2 < 3) \equiv \text{adev\u0103rat}$
- 3.3:  $p \leftarrow i+1 \equiv 4$
- 3.4:  $p \leftarrow p-1 \equiv 3, a[3] \leftarrow a[2] \equiv 5$
- 3.5:  $(p > j+1) \equiv (3 > 2) \equiv \text{adev\u0103rat}$
- 3.4:  $p \leftarrow p-1 \equiv 2, a[2] \leftarrow a[1] \equiv 3$
- 3.5:  $(p > j+1) \equiv (2 > 2) \equiv \text{fals}$
- 3.6:  $a[1] \leftarrow 2$
- 3.7:  $i \leftarrow i+1 \equiv 4$

În tabloul a se afl\u0103 \u00een acest moment (2, 3, 5, -), deci o ordonare cresc\u0103toare.

pas 3:  $(i \leq n) \equiv (4 \leq 4) \equiv \text{adev\u0103rat}$

- 3.1:  $x \leftarrow 3, j \leftarrow 1$
- 3.2:  $(j < i) \equiv (1 < 4) \equiv \text{adev\u0103rat}$ 
  - 3.2.1:  $(x < a[j]) \equiv (3 < 2) \equiv \text{fals}$
  - 3.2.2:  $j \leftarrow j+1 \equiv 2$ 
    - 3.2.1:  $(x < a[j]) \equiv (3 < 3) \equiv \text{fals}$
    - 3.2.2:  $j \leftarrow j+1 \equiv 3$ 
      - 3.2.1:  $(x < a[j]) \equiv (3 < 5) \equiv \text{adev\u0103rat}$
- 3.3:  $p \leftarrow i+1 \equiv 5$
- 3.4:  $p \leftarrow p-1 \equiv 4, a[4] \leftarrow a[3] \equiv 5$
- 3.5:  $(p > j+1) \equiv (4 > 4) \equiv \text{fals}$
- 3.6:  $a[3] \leftarrow 3$
- 3.7:  $i \leftarrow i+1 \equiv 5$

pas 3:  $(i \leq n) \equiv (5 \leq 4) \equiv \text{fals}$

Se scrie tabloul ob\u021binut : (2, 3, 3, 5).

Algoritmul `ord_3` func\u021bioneaz\u0103 \u00een felul urm\u0103tor:

Numerele se introduc pe r\u00e2nd. Dup\u0103 introducerea unei variabile, se parcurge tabloul \u0219i se g\u0103se\u0219te pozi\u021bia unde trebuie pus acest num\u0103r pentru a avea o ordonare cresc\u0103toare.

Ciclurile algoritmului sînt:

pas 3: cu contor  $i$ , parcurs în  $n-1$  pași;

pas 3.2: buclă cu număr necunoscut de pași deși conține un contor  $j$ ; ciclul este parcurs de maxim  $i-1$  ori;

pașii 3.4 - 3.5: cu contor  $p$ , parcurs în  $i-j+1$  pași. Variabila contor  $p$  descrește 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 și descifrarea amănunțită a fiecărui pas (rolul său, importanța operațiilor de condiție, scrierea cu operații de decizie alternativă, etc.



## EXERCITII

1. Se dau numerele întregi  $a$  și  $b$ . Să se scrie o fracție ireductibilă  $x/y$  cu proprietatea  $x/y=a/b$ .
2. Se dă un număr  $n$ . Să se afle toți 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  $a$  și  $b$  prime între ele ( $\text{cmmdc}(a,b)=1$ ). Să se determine două numere întregi  $x, y$  astfel încît  $a*x+b*y=1$ .
6. Se dau două numere întregi  $a$  și  $b$ . Să se găsească două numere întregi  $x$  și  $y$  astfel încît  $a*x+b*y=d$  unde  $d = \text{cmmdc}(a,b)$ .
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 și 2.

8. Dintr-un tablou **a** cu o dimensiune să se obțină un tablou **b** în care toate elementele lui **a** apar o singură dată.
9. În aceleași condiții, să se obțină 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 operații:  
 - permutarea a două linii între ele (element cu element);  
 - permutarea a două coloane între ele.  
 Să se transforme tabloul în așa fel încît elementele de pe diagonala principală să fie ordonate descrescător.
11. Se dă o propoziție. Să se găsească litera care se repetă de cele mai multe ori.
12. Se dă o propoziție avînd cuvintele separate prin spațiu și încheiată cu punct. Să se determine cîte cuvinte are propoziția.  
 Cîte din ele sînt formate din două litere ?  
 Care este cel mai lung cuvînt ?



Fiecare ascultă numai ce înțelege.

(Goethe)

## Anexa 1

### Scrierea algoritmilor în limbajul BASIC

Programele au fost scrise în varianta Q-BASIC și 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 "Alegeți 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 maxmin\_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 "Dat̃i 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 "Dat̃i 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 "Dat̃i cele doua numere": INPUT a,b
15 IF a*b=0 THEN PRINT 'Eroare de date": GOTŌ 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 THEN 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 știință;  
dar cine crede tot ce vorbește ?  
(Gr. C. Moisil)*

## Anexa 2

### Scrierea algoritmilor în limbajul PASCAL

**Observație:** Programele au fost scrise în varianta PASCAL 5.5 și 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);
end.
```

```
writeln(a,b);
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
 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
 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
end.

```

---

```

program joc_2;
var i:integer;
 x:string;
begin
 writeln('Alegeti un nume din lista A. El poate fi ghicit din 4 intrebari.');
```

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
 end;
 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;
 end
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_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 decat ',x);
writeln('Sint ',egal,' elemente egale cu ',x);
writeln('Sint ',mic,' elemente mai mici decat ',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 pozitiiile:');
 for i:=1 to kmax do write(maxim[i],' '); writeln;
 writeln('Cel mai mic numar este ',min,
 '. El apare in tablou pe pozitiiile: ');
 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
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
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 ;
 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;
 end;
end.

```

```
 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 prezentați

| Nume      | Algoritm (pag) | Program Basic | Program Pascal |
|-----------|----------------|---------------|----------------|
| comp_1    | 99             | 152           | 175            |
| comp_2    | 102            | 153           | 175            |
| comp_3    | 104            | 153           | 176            |
| comp_4    | 104            | 154           | 176            |
| comp_5    | 120            | 156           | 180            |
| div_1     | 73             | 147           | 167            |
| div_2     | 75             | 148           | 167            |
| ecuație_1 | 76             | 148           | 168            |
| ecuație_2 | 78             | 149           | 168            |
| euclid_1  | 126            | 158           | 183            |
| euclid_2  | 128            | 159           | 183            |
| euclid_3  | 130            | 159           | 184            |
| euclid_4  | 131            | 159           | 184            |
| joc_1     | 79             | 149           | 169            |
| joc_2     | 84             | 150           | 171            |
| joc_3     | 85             | 151           | 172            |
| max_1     | 67             | 147           | 165            |
| max_2     | 68             | 147           | 166            |
| max_3     | 69             | 147           | 166            |
| max_4     | 72             | 147           | 166            |
| maxim_1   | 105            | 154           | 177            |
| maxim_2   | 108            | 154           | 178            |
| maxmin_1  | 109            | 155           | 178            |
| maxmin_2  | 110            | 155           | 179            |
| maxmin_3  | 120            | 157           | 181            |

| Nume      | Aloritm(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  
circumferința 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 Universității București, 1987.
3. C. Calude - *Teoria algoritmilor*, Tipografia Universității București, 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 și programe Fortran*, Ed. Did. și Ped., 1978.
7. I. Tomescu, A. Leu - *Matematica aplicată în tehnica de calcul*, Manual clasa X, Ed. Did. și Ped., 1980.
8. *Communication of the Association Computing Machinery*, 1959 - 1986.
9. *Gazeta de Informatică*, nr. 1,2 (1991), 1-11 (1992).

*Au apărut:*

Ion Diamandi: **Cum să realizăm jocuri pe calculator**

Luminița State: **Hello, BASIC** (ed. I și II)

Adrian Atanasiu: **Cum se scrie un algoritm? Simplu**

*Vor apărea (toamna 1993):*

Ion Diamandi: **Calculatorul, coleg de bancă**

Marian Gheorghe: **Cine ești tu, BASIC ?**

Răzvan Andonie, Ilie Gârbacea: **Algoritmi fundamentali în C++**

Lucrările se pot obține prin poștă (plata ramburs, la primire), scriind pe adresa:

Editura AGNI, C.P. 30-107 București.

Model pentru comanda dvs:

Subsemnatul..... str..... nr..... bl..... sc... apt.....  
localitatea.....județul.....cod.....  
doresc următoarele lucrări:  
Titlul..... Nr.exemplare.....  
Titlul..... Nr.exemplare.....  
etc.

---

Recomandăm tuturor celor interesați de domeniu excelenta **Gazeta de informatică**  
editată de Editura Libris, Cluj, str. Universității 8, tel. 095/ 112422.



### *Despre carte :*

Ea conține 50 de algoritmi fundamentali, cu programele respective, în limbajele BASIC și PASCAL.

Cartea vine în sprijinul celor ce doresc să învețe să programeze. Poate fi un material didactic prețios pentru informatica predată în clasele VIII-XII.

### *Cum se scrie un algoritm ? Simplu*

### *Despre autor :*

Dl. **Adrian Atanasiu** este conferențiar la Facultatea de Matematică (Universitatea București). Predă din 1977 cursuri de scriere a compilatoarelor, bazele informaticii, coduri și criptografie.

Este redactor al Gazetei de Informatică, membru în juriul mai multor concursuri de informatică pentru elevi.