TUDOR SORIN

MANUAL PENTRU
CLASA A IX-A

INFORMATICA

PROFILUL REAL INTENSIV
VARIANTA C++

RESPECTĂ NOUA PROGRAMĂ
VALABILĂ ÎNCEPÂND CU ANUL 2004

APROBAT MEC cu ordinul nr. 4188/02.07.2004

CAPITOLUL 1

Algoritmi

1.1. Noțiuni generale

Noțiunea de algoritm este primară, nu se definește, întocmai ca și noțiunea de mulțime în matematică. Cu toate acestea, ca și mulțimea, algoritmul poate fi descris. *În esență, este vorba de o succesiune de etape care se pot aplica mecanic în vederea obținerii unui anumit rezultat.*

În activitatea cotidiană întâlnim la tot pasul algoritmi. Vom da câteva exemple.

1. *Algoritmul prin care o persoană dă un telefon.* Persoana ridică receptorul, așteaptă tonul. Dacă nu vine tonul, închide telefonul, apoi îl redeschide. După apariția tonului formează numărul, etc. Nu îmi propun să descriu amănunțit algoritmul prin care se dă un telefon pentru că este mult prea cunoscut, dar atrag atenția asupra faptului că se poate descrie foarte bine, astfel încât persoana poate executa mecanic operațiile necesare.

2. *Algoritmul prin care o persoană poate găti un anumit fel de mâncare.* Persoana are la dispoziție făină, zahăr, ouă etc. pe care le combină în anumite proporții, după care amestecul obținut se fierbe (se prăjește sau se pune în cuptor) un anumit interval de timp. Produsului obținut i se mai poate adăuga câte ceva, după care este din nou fiert (prăjit) un interval de timp, după care se obține produsul finit, adică mâncarea dorită.

3. *Algoritmul prin care se adună două fracții.* Cele două fracții se aduc la același numitor, se fac înmulțirile, adunările, apoi fracția este simplificată.

Oricare din algoritmii de mai sus poate fi descris în termeni precisi, astfel încât cel care-l execută poate efectua operațiile fără să fie nevoit să gândească ce are de făcut la un anumit moment.

O analiză sumară a exemplelor ne conduce la următoarele observații:

➢ *În orice algoritm se pornește de la ceva și se dorește obținerea unui anumit rezultat.*

- Dăm un telefon pentru ca un anumit mesaj să ajungă la destinație. Se pornește de la un mesaj și se dorește ca acesta să ajungă la o anumită persoană.
- Dacă dorim să gătim un anumit fel de mâncare, pornim de la anumite produse și obținem mâncarea solicitată.
- Dacă adunăm două fracții, pornim de la cele două fracții și obținem suma lor.

➢ *În orice algoritm se operează cu anumite "obiecte" asupra cărora sunt permise anumite operații.*

- Algoritmul prin care dăm un telefon operează cu telefonul. Operațiile permise sunt deschiderea, închiderea telefonului, formarea numărului etc.
- Algoritmul prin care se obține un anumit fel de mâncare operează cu farfurii, tăvi, aragaz, alimente, etc. Operațiile permise sunt amestecarea, fierberea, prăjirea alimentelor.
- Algoritmul prin care se adună două fracții operează cu valori numerice. Operațiile sunt cele descrise de regulile matematice.

➢ *În cele mai multe cazuri cel care elaborează algoritmul este diferit de executant.*

- Pentru a putea fi aplicat algoritmul prin care se dă un telefon a fost necesară inventarea telefonului, gândirea modului în care cineva dă ușor un telefon. O mulțime de personalități au gândit toate acestea, dar algoritmul poate fi aplicat de orice persoană care știe cifrele între **0** și **9**.
- Pentru a putea fi aplicat algoritmul prin care se obține un anumit fel de mâncare au fost făcute numeroase experimente care au condus până la urmă la o rețetă. Poate găti orice persoană care știe să citească și să folosească aragazul.
- Pentru a putea aduna două fracții a fost necesar ca matematica să se dezvolte suficient de mult. Să nu uităm că oamenii au lucrat mult timp doar cu numere naturale...

Din cele de mai sus, rezultă că noțiunea de algoritm este extrem de generală, cu ea ne întâlnim tot timpul. *În această carte ne ocupăm numai de elaborarea algoritmilor pentru programarea calculatoarelor.* Aici "executantul" este calculatorul, iar cel care elaborează algoritmul poartă numele de "programator". Calculatorul doar execută instrucțiuni, nu gândește, dar viteza de executare a instrucțiunilor este foarte mare, imposibil de atins de om. Frecvent, mai intervine o persoană, de cele mai multe ori diferită de programator, numită "utilizator". Ea este cea care utilizează programul obținut și beneficiază de avantajele lui. De cele mai multe ori, o astfel de persoană are o pregătire minimă de specialitate în informatică. Din păcate, se face deseori confuzia între programator și utilizator.

1.2. Enunțul unei probleme, date de intrare și de ieșire, etapele rezolvării unei probleme

Se consideră funcția:

$$f:\Re\to\Re,\quad f(x)=\begin{cases}x^2-2, & x<0;\\ 3, & x=0;\\ x+2, & x>0.\end{cases}$$

Se cere să se calculeze `f(x)` pentru `10` valori reale ale lui `x`. De exemplu: `-3.1, 7, 0, 2.23,` ș.a.m.d.

Considerăm prima valoare a lui `x`, și anume `-3.1`. Observăm că această valoare este mai mică decât `0`. Calculăm `(-3.1)^2-2`. A doua valoare pentru care trebuie calculată funcția este `7`. Pentru că ea este mai mare decât `0`, calculăm x+2, adică `7+2`. A treia valoare este `0`. Funcția ia valoarea `3`. Repetăm calculul pentru cele `7` valori rămase. Această modalitate de calcul este plicticoasă și cere mult timp. Pentru a rezolva astfel de probleme -și nu numai de tipul acesteia- a fost inventat calculatorul. Acesta va efectua calculele în locul nostru. Observați faptul că *pentru efectuarea calculelor, calculatorul trebuie să ia anumite decizii automat*. Astfel, în funcție de valoarea lui `x` (negativă, zero, sau mai mare decât 0) acesta va decide ce expresie calculează pentru a obține `f(x)`. Dacă calculele elementare pot fi făcute de un simplu calculator de buzunar, neprogramabil, deciziile automate le poate lua numai un calculator programabil. Evident, deciziile se iau în urma aplicării algoritmului.

Calculatorul rezolvă o problemă atunci când execută un anumit program, corespunzător problemei. În esență, programul este alcătuit din comenzi (instrucțiuni) pe care calculatorul le execută. *Programul se obține prin codificarea algoritmilor într-un limbaj de programare.*

În continuare, prezentăm în linii mari etapele obținerii unui program.

1. Identificarea datelor de intrare și a celor de ieșire

Am văzut faptul că în orice algoritm se pornește de la ceva și se urmărește un anumit rezultat. Cu alte cuvinte, trebuie să ne fie clar de la ce plecăm și ce vrem să obținem. Aceasta înseamnă că trebuie să cunoaștem datele de intrare -de la ele plecăm- si datele de ieșire -pe ele trebuie să le obținem. Pentru exemplul considerat, datele de intrare sunt `10` valori reale x_1, x_2, ..., x_{10}. Datele de ieșire sunt cele `10` valori calculate `f(x1), f(x2), ..., f(x10)`.

Observație. Algoritmii operează cu date de intrare și de ieșire, chiar și atunci când acest fapt nu este atât de evident. Să presupunem că jucăm un joc pe calculator. Aceasta are loc sub controlul unui anumit program, care a fost obținut în urma codificării unui algoritm. O dată de intrare poate fi apăsarea unei taste, un clic al *mouse*-ului etc. O dată de ieșire poate fi o anumită imagine sau o succesiune de imagini care creează impresia că un obiect se deplasează, un anumit sunet (și acestea sunt codificate numeric), etc.

2. Elaborarea algoritmului de rezolvare a problemei.

În linii mari, algoritmul specifică operațiile pe care le "are de făcut" calculatorul pentru ca, pornind de la datele de intrare, să obțină datele de ieșire. Iată, de exemplu, cum arată algoritmul simplificat de rezolvare a problemei propuse:

Se parcurg următoarele etape, de **10** ori:

- se citește valoarea lui **x**;
- în funcție de valoarea proprie a lui **x**, se procedează astfel:
  - dacă este negativă, se calculează **f=x²-2**;
  - dacă este **0**, valoarea funcției este **3**;
  - dacă este pozitivă, se calculează **f=x+2**;
- se scrie valoarea calculată pentru **f**.

Observați faptul că algoritmul a fost descris în limbaj natural. În practică se folosesc limbaje de tip pseudocod sau, din ce în ce mai rar, scheme logice.

Ce este un limbaj de tip pseudocod? După cum știm, programele pentru calculator sunt scrise în anumite limbaje: **Pascal**, **C++**, **Fortran**, **Cobol** etc. Pentru a scrie programele într-un limbaj de programare este necesar să respectăm anumite reguli specifice limbajului. Atunci când elaborăm algoritmul care stă la baza programului este greu să avem în vedere și regulile specifice limbajului. Trebuie să ne concentrăm asupra problemei, nu asupra unor detalii. Atunci va trebui să folosim un limbaj de tip pseudocod. *Adică un limbaj care nu are multe reguli, dar care seamănă cu orice limbaj de programare.* Un algoritm redactat în pseudocod nu poate fi rulat pe calculator, este necesară conversia sa în limbajul de programare dorit. Însă conversia este aproape mecanică, pentru că atunci nu suntem concentrați asupra algoritmului. Atragem atenția asupra faptului că un adevărat limbaj de tip pseudocod permite *conversia cu ușurință a algoritmului în orice limbaj de programare.* Există multe limbaje de tip pseudocod, aproape orice persoană poate să creeze unul. *În acest capitol folosim un limbaj de tip pseudocod în limba română, limbaj utilizat în prezent în cadrul examenului de bacalaureat.*

Ca și limbajele de tip pseudocod, schemele logice permit redactarea algoritmilor. Ele au fost mult folosite în trecut, dar astăzi se utilizează din ce în ce mai rar. Motivul? Ocupă mult spațiu pe hârtie și creează neplăceri privind reprezentarea grafică. Cu toate acestea sunt considerate intuitive și mulți profesori le utilizează.

3. Transpunerea algoritmului într-un limbaj de programare (**Pascal, C++**)

În această etapă se operează aproape mecanic. Odată cunoscut un limbaj de programare, transpunerea algoritmului nu este o problemă dificilă. Rețineți: *nu este greu să învețI un anumit limbaj, este greu să știi să elaborezi algoritmi.*

De fapt, algoritmul poate fi codificat direct în limbajul de programare dorit. Cu timpul, când veți căpăta suficientă experiență și dacă problema nu este dificilă chiar așa veți proceda.

4. Testarea programului și corectarea sa până când "funcționează" corect.

Atunci când programul este complex, este aproape imposibil să fie scris corect de la început. Din acest motiv este necesară faza de mai sus. Nu uitați, calculatorul este cel mai bun corector, hârtia suportă orice greșeală.

1.3 Noțiunea de algoritm, caracteristici

Noțiunea de algoritm a mai fost prezentată în primul paragraf al acestui capitol. Aici o reamintim, o particularizăm pentru programarea calculatoarelor și evidențiem caracteristicile sale. *Prin algoritm înțelegem o succesiune de etape care se pot aplica mecanic pentru ca, pornind de la datele de intrare, să se obțină datele de ieșire.* Rețineți: pentru orice algoritm trebuie precizat în mod clar care sunt datele de intrare și care sunt cele de ieșire. Dacă această precizare nu a fost făcută, nu se mai poate vorbi de algoritm. Iată câteva caracteristici ale algoritmilor.

1. Finititudine - *este proprietatea algoritmilor de a furniza rezultatele într-un timp finit.* Nu trebuie înțeles de aici că dacă un algoritm furnizează rezultatele în timp finit este neapărat bun. El trebuie să fie și eficient, adică să alegem calea cea mai simplă de rezolvare, înțelegând prin aceasta, cea care presupune un efort de calcul cât mai mic.

Exemplu: *Se cere să se elaboreze algoritmul prin care se listează primele 100 pătrate perfecte.* O primă idee ar fi să testăm care număr (începând cu **1**, continuând cu **2**, **3**, etc.) este pătrat perfect. Dacă acesta îndeplinește condiția este tipărit, altfel se trece mai departe. Algoritmul se termină atunci când am listat **100** de pătrate perfecte. Altfel: se tipăresc valorile 1^2, 2^2, ..., 100^2. E mai simplu, nu? Să vedem de ce. În primul caz se analizează primele **10000** numere naturale (pentru că $100^2=10000$)

și se tipăresc cele care sunt pătrate perfecte. În al doilea caz se efectuează doar **100** de înmulțiri. Veți spune că nu contează, calculatorul face calculele extrem de rapid. Nu-i așa! Pentru probleme complexe, un algoritm performant rulează cu mult mai repede decât unul care nu îndeplinește această condiție. *Aceasta înseamnă că, atunci când avem de elaborat un algoritm, nu ne vom opri la prima soluție găsită.* Referitor la această proprietate, se impun anumite precizări. Există probleme pentru care nu se cunosc algoritmi de rezolvare rapizi. *Pentru anumite seturi de date de intrare, este necesar un timp de calcul de ordinul sutelor sau chiar miilor de ani și aceasta pe calculatoarele ultramoderne.* Din acest motiv, în practică se consideră că pentru aceste probleme nu se cunosc algoritmi de rezolvare, chiar dacă timpul de lucru este finit.

2. Claritatea - *este proprietatea algoritmilor prin care procesul de calcul este descris precis, fără ambiguități.*

3. Generalitatea - *este proprietatea algoritmilor de a rezolva o întreagă clasă de probleme.* Să considerăm problema pusă în acest paragraf. De fapt, funcția se calculează pentru *oricare* **10** valori reale. Mai general ar fi fost să calculăm valorile pe care le ia funcția pentru oricare **n** valori reale, cu **n** număr natural citit. Mai mult, se poate obține un algoritm care să primească drept date de intrare atât numărul de valori, valorile propriu-zise, cât și funcția ale cărei valori trebuie calculate.

1.4. Obiectele cu care lucrează algoritmii și operații permise

În linii mari, algoritmii lucrează cu două tipuri de obiecte: date și variabile. Cu acestea sunt permise anumite operații. Precizăm că operațiile care vor fi efectuate sunt exemplificate în limbaj de tip pseudocod.

1.4.1. Date

Așa cum am văzut, orice algoritm pornește de la anumite date de intrare, le prelucrează, iar în final obține date de ieșire. În exemplul prezentat, datele de intrare sunt reprezentate de cele **10** valori ale lui **x**, iar datele de ieșire sunt cele **10** valori ale lui **f(x)**.

Datele pot fi clasificate după tipul lor:

- întregi;
- reale;
- logice;
- șir de caractere.

Datele din primele două tipuri poartă numele de date numerice.

Datele întregi sunt numere aparținând mulțimii numerelor întregi. Exemple: **100, -6700, +122**.

Datele reale sunt numere cu zecimale. La o astfel de dată, în loc de virgulă se folosește punctul. Exemplu: în loc de **3,25** se scrie **3.25**. După cum știm din matematică, mulțimea numerelor reale este o reuniune între mulțimea numerelor raționale și mulțimea numerelor iraționale. Nici un calculator din lume nu poate reține un număr irațional, întrucît acesta are o infinitate de zecimale. Din acest motiv, în informatică, printr-o dată reală înțelegem o valoare cu un număr finit de zecimale.

Datele logice au numai două valori: **TRUE** (adevărat) și **FALSE** (fals).

Datele de tip șir de caractere sunt reprezentate de șiruri de caractere cuprinse între apostrof. Exemple: **'un text'**, **'facem reforma'**.

Aici este momentul să vorbim despre o categorie aparte de date și anume constantele. Constantele sunt date care nu se modifică pe parcursul aplicării algoritmului. Acestea pot fi date de oricare dintre tipurile precizate. *Ele se caracterizează prin faptul că sunt utilizate în algoritm, fără a fi citite sau obținute din calcule.* Se folosesc, de exemplu, în calculele care se fac sau sunt mesaje care trebuie să apară la fiecare rulare a programului rezultat în urma algoritmului, etc.

1.4.2. Variabile

În problema analizată aveam de citit **10** valori de intrare. În matematică acestea ar fi notate x_1, x_2, ..., x_{10}. În prelucrare, pentru orice dată de intrare de acest tip se fac aceleași operații. Este lipsit de sens să explicăm ce facem cu x_1 pentru ca apoi să explicăm ce facem cu x_2 etc. Din acest motiv, în algoritm se prezintă ce se face cu un anume **x**, oricare ar fi el dintre cele **10**. De asemenea, nu are rost să precizăm că se tipărește **f(**x_1**)**, apoi **f(**x_2**)** ș.a.m.d. Se notează pur și simplu că se tipărește **f**.

Până în acest moment avem o formalizare matematică comună. De aici s-a și pornit atunci când s-a ajuns la variabile. Pentru algoritmul nostru **x** și **f** sunt variabile. *Ne imaginăm variabilele ca pe niște "cutiuțe" care reținu date.* Fiecare variabilă are un nume.

Exemple:

- Variabila **a** reține data întreagă **0**.

- Variabila **a** reține data întreagă **6**.

- Variabila **b** reține data de tip șir: **'un mesaj'**.

b

O variabilă poate reține date numai de un tip anume. Astfel avem variabile care pot reține date întregi, variabile care pot reține date reale, variabile care pot reține date logice și variabile care pot reține date de tip șir de caractere.

De aici rezultă o caracteristică fundamentală a variabilelor: tipul lor. Acesta determină natura datelor care pot fi reținute de variabila respectivă. Astfel avem:

- variabile de tip întreg - le vom nota în pseudocod cu **întreg**;
- variabile de tip real - notate **real;**
- variabile de tip logic - notate **logic;**
- variabile de tip șir - notate **șir.**

Facem următoarea convenție: pentru ca un algoritm să poată folosi o variabilă, aceasta trebuie declarată - adică anunțată. Iată cum arată declarațiile variabilelor **a** și **b**:

```
întreg a
șir b;
```

Dacă trebuie să declarăm mai multe variabile de același tip procedăm ca în exemplul următor:

```
întreg c, d
logic e, f.
```

✓ Cu toate că o variabilă are un nume unic, conținutul ei poate fi diferit de la un moment la altul, pe parcursul executării programului. De aici provine denumirea de variabilă.

Dar dacă, în mod abstract, este posibil să notăm o intrare cu **x** și o ieșire cu **f**, cum este posibil să folosim această abstractizare în momentul în care algoritmul este transpus într-un limbaj de programare? Practic, în memoria calculatorului se rezervă pentru fiecare variabilă un spațiu. Aceasta înseamnă că am rezervat un singur spațiu pentru **x** și un singur spațiu pentru **f**.

1.4.3. Expresii

În scopul efectuării calculelor, sau a luării anumitor decizii algoritmii folosesc *expresii*. O expresie este alcătuită din unul sau mai mulți operanzi legați între ei prin operatori.

- Operanzii pot fi constante și variabile.
- Operatorii au rolul de a preciza operațiile care se efectuează.

În linii mari, expresiile sunt de două feluri:

A) Expresii aritmetice

B) Expresii logice.

A) Expresii aritmetice. Operanzii sunt constante sau variabile de tip întreg sau real. Mai des întâlniți sunt operatorii:

- pentru adunare folosim operatorul '+';
- pentru scădere folosim operatorul '-';
- pentru înmulțire folosim operatorul '*';
- pentru împărțire folosim operatorul '/'.

Exemplu: dacă **a** și **b** sunt două variabile de tip întreg care reține **3**, respectiv **7**, atunci expresia **a+3*b** ia valoarea **3+3x7**, adică **24**.

✓ În pseudocod putem utiliza ca operator orice simbol cu semnificație matematică. Exemple: $\sqrt{\ }$ pentru radical, [ ] pentru parte întreagă etc.

Exemplu: dacă **y** este o valoare întreagă care reține **9**, expresia $\sqrt{y}+\left[\frac{y}{2}\right]$ ia valoarea $\sqrt{9}+\left[\frac{9}{2}\right]$, adică $3+[4.5]$, adică **7**.

B) Expresii logice.

Anumiți operatori, deși au ca operanzi variabile, constante sau expresii aritmetice produc ca rezultat valori logice: **true**, **false**.

Din prima categorie fac parte operatorii de comparare:

- pentru egalitate folosim operatorul '='
- pentru inegalitate folosim operatorul '≠'
- pentru mai mare folosim operatorul '>'
- pentru mai mare sau egal folosim operatorul '≥'
- pentru mai mic folosim operatorul '<'
- pentru mai mic sau egal folosim operatorul '≤'.

Exemple: Variabila **a** reține 2, iar variabila **b** reține 5. Atunci:

a<b ia valoarea **true**;
a>b ia valoarea **false**;
a=b ia valoarea **false**;
a≠b ia valoarea **true**.
a≥2 ia valoarea **true;**
b≤6 ia valoarea **true.**

Alți operatori au ca operanzi variabile, constante, expresii logice și produc ca rezultat valori logice: **true**, **false**.

- operatorul **SAU** acționează asupra a doi operanzi logici și dacă *cel puțin unul din ei este* **true**, rezultatul este **true**, contrar rezultatul este **false**.
- operatorul **SI** acționează asupra a doi operanzi logici și dacă *ambii* sunt **true**, rezultatul este **true**, contrar rezultatul este false.
- operatorul **NOT** acționează asupra unui operand logic și dacă operandul este **true**, rezultatul este **false**, contrar rezultatul este **false**.

Exemple:

false SAU true ia valoarea **true**;
false SAU false ia valoarea **false**;
false SI true ia valoarea **false**;
true SI true ia valoarea **true**;
NOT false ia valoarea **true**;
NOT true ia valoarea **false.**

Relații foarte importante:

NOT(a>b) ⇔ a≤b. Se poate gândi așa: dacă **a** nu este mai mare ca **b**, atunci **a** este mai mic sau egal cu **b** și este corect. În realitate, calculatorul lucrează cu expresii logice.

Între **a** și **b** pot exista 3 relații:

1. **a<b.** Atunci **a>b** ia valoarea **false**; **NOT(false)=true.** Pe de altă parte, în aceleași condiții, **a≤b** ia valoarea **true**. Avem egalitate.

2. **a>b.** Atunci **a>b** ia valoarea **true**; **NOT(true)=false.** Pe de altă parte, în aceleași condiții, **a≤b** ia valoarea **false**. Avem egalitate.

3. **a=b.** Atunci **a>b** ia valoarea **false**; **NOT(false)=true.** Pe de altă parte, în aceleași condiții, **a≤b** ia valoarea **true**. Avem egalitate.

De asemenea, mai avem relațiile de mai jos, care se demonstrează în mod asemănător:

NOT(a<b) ⇔ a≥b
NOT(a≤b) ⇔ a>b
NOT(a≥b) ⇔ a<b

Mai avem relațiile lui *de Morgan* (**a** și **b** sunt cele două valori logice).

1. NOT(a SI b) =NOT a SAU NOT b;
2. NOT(a SAU b) =NOT a SI NOT b;

Pentru demonstrarea acestor relații se iau în considerare toate valorile pe care le pot lua valorile logice **a** și **b**. Demonstrăm prima relație.

a) a=true, b=true. a SI b =true. NOT (a SI b)=false. Pe de altă parte: **NOT a= false, NOT b= false; false SAU false =false.**

b) a=true, b=false. a SI b =false. NOT (a SI b)=true. Pe de altă parte: **NOT a= false, NOT b= true; false SAU true =true.**

c) a=false, b=false. a SI b =false. NOT (a SI b)=true. Pe de altă parte: **NOT a= true, NOT b= true; true SAU true =true.**

d) a=false, b=true. a SI b =false. NOT (a SI b)=true. Pe de altă parte: **NOT a=true, NOT b= false; true SAU false =true.**

1.5. Operațiile pe care le efectuează un algoritm

În linii mari, operațiile care se fac într-un algoritm se împart în trei mari categorii:

- operații de intrare / ieșire;
- operații de atribuire;
- operații de decizie.

În trecut, se mai folosea o operație și anume cea de salt.

1.5.1. Operații de intrare / ieșire

Am arătat că orice algoritm lucrează cu date de intrare și ieșire. Acestea pot fi citite sau scrise.

Prin operația de intrare (de citire) se înțelege preluarea unei date de la un dispozitiv de intrare către memoria internă a calculatorului, în zona de memorie rezervată pentru aceasta, adică în variabilă. Dispozitivele de intrare pot fi tastatura (cel mai des utilizată); o unitate de dischetă; o unitate de disc, etc.

În pseudocod, pentru citire vom folosi operația `Citeşte`.

Prin operația de ieșire (scriere) se înțelege preluarea unei date din memoria internă -adică dintr-o variabilă- și transferul ei către un dispozitiv de ieșire. Dispozitivele de ieșire pot fi monitorul, o unitate de disc, imprimanta, etc.

Pentru scriere vom folosi operația **`Scrie`**.

Deocamdată vom presupune că dispozitivul de intrare este tastatura, iar cel de ieșire este monitorul. Să analizăm algoritmul de mai jos -scris în pseudocod- care citește un număr întreg și îl tipărește:

```
întreg a
Citeşte  a
Scrie a
```

Mai întâi am declarat variabila **a**, de tip întreg - adică are posibilitatea de a reține numere întregi. Urmează să se efectueze operația de citire a variabilei **a**. Aceasta înseamnă că se așteaptă introducerea de la tastatură a datei întregi. În cazul în care introducem numărul **10**, variabila **a** va reține **10**, dacă introducem numărul **176**, variabila **a** va reține **176**. Să presupunem că am citit **176**. Iată cum ne imaginăm variabila **a**:

a

La scriere, pe monitor apare numărul **176** - conținutul variabilei **a**.

Observații:

- După scriere, conținutul variabilei rămâne nemodificat.

Fie secvența de mai jos și introducem de la tastatură **17**. Pe monitor apare de două ori numărul **17**.

```
întreg a
Citește a
Scrie a
Scrie a
```

- La o nouă citire, conținutul vechi al variabilei se pierde.

Fie secvența de mai jos. Presupunem că introducem de la tastatură **10, 12** (în această ordine). Atunci variabila **a** va reține **12**, deci pe monitor apare **12.**

```
întreg a;
Citește a
Citește a
Scrie a;
```

Se pot citi mai multe variabile cu o singură operație **Citește** și se pot tipări valorile reținute de mai multe variabile cu o singură operație **Scrie.**

Exemplu: Fie secvența:

```
real a,b,c;
Citește a,b,c
Scrie a,b,c
```

Dacă introducem **1.2, -1.3, 12.8,** atunci **a** reține **1.2, b** reține **-1.3, c** reține **12.8**. Pe monitor se tipărește **1.2 -1.3 12.8**.

1.5.2. Atribuiri

Prin operația de atribuire se reține o anumită dată într-o variabilă. Ea are mai multe forme, pe care le vom prezenta pe rând.

Forma 1.

v←dată

Semnificația este următoarea:

- v este numele unei variabile de un tip oarecare.
- ← notația pentru operația de atribuire;
- dată - o valoare de un tip oarecare.

Cu o singură excepție, tipul variabilei trebuie să coincidă cu tipul valorii atribuite.

Exemple:

```
întreg a;
a←10;
```

Variabila **a** va reține **10**.

```
real b;
b←-7.25
```

Variabila **b** reține **-7.25**.

```
șir c;
c←'latina si filosofie'
```

Variabila **c** reține valoarea de tip șir **'latina si filosofie'**.

Excepția este următoarea: *unei variabile de tip real i se poate atribui o dată de tip întreg.*

```
real d;
d←7
```

Forma 2.

v1←v2

unde, **v1** și **v2** sunt nume de variabile. Cu o excepție, tipul celor două variabile trebuie să coincidă. Efectul este că variabila **v1** va reține conținutul variabilei **v2**. După aplicarea acestei operații, conținutul variabilei **v2** rămâne nemodificat, iar conținutul inițial al variabilei **v1** se pierde.

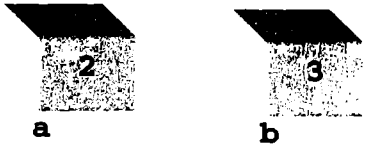
Exemplul 1.

Fie **a** o variabilă de tip **întreg** care reține valoarea **2**. Variabila **b** este de același tip cu variabila **a** și reține numărul **3**. Considerăm că se efectuează atribuirea **a←b**. Conținutul inițial al celor două variabile este:

După atribuire, conținutul celor două variabile va fi:

Exemplul 2.

La fel ca în exemplul 1. Considerăm atribuirea **b←a**. Conținutul inițial al celor două variabile este:



După atribuire, conținutul va fi:

Concluzie. *Nu este indiferent modul de scriere al variabilelor în cadrul atribuirii.* Atribuirea **a←b** nu este identică cu atribuirea **b←a**. Este adevărat că, după atribuire, cele două variabile vor avea același conținut. Însă este important ce conținut (al lui **b** după prima atribuire, al lui **a** după a doua). Aici începătorii greșesc deseori. *Atenție!*

Forma 3.

v←expresie

Inițial se evaluează expresia, iar valoarea obținută este atribuită variabilei **v**. Considerăm că dacă cel puțin unul din operanzi este real, tipul expresiei este real si poate fi atribuit doar unei variabile de tip real, iar dacă toți operanzii sunt întregi, tipul expresiei este întreg și poate fi atribuit unei variabile de tip întreg sau real. Facem convenția ca operatorul '**/**' va da întotdeauna un rezultat real, chiar dacă ambii operanzi sunt întregi.

Observații. După atribuire, variabilele care sunt operanzi rămân nemodificate în ce privește conținutul. Excepție face, eventual, variabila **v** în cazul în care figurează și ca operand în expresie. De asemenea, primele două forme sunt particularizări ale acesteia.

Exemplul 1.

Fie **v** o variabilă de tip întreg care conține numărul **7**. Considerăm că se face atribuirea: **v←v+1**. Conținutul variabilei **v** înainte de atribuire este:

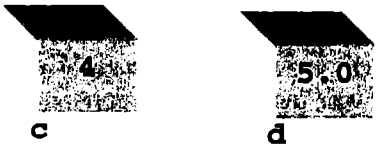
După atribuire acesta devine:

Explicație. La pasul 1 a fost evaluată expresia **v+1**. În urma evaluării s-a obținut valoarea **8** (de tip **întreg**). Această valoare a fost atribuită variabilei **v**. Evident, vechiul conținut al lui **v** s-a pierdut.

Exemplul 2.

Fie **c** o variabilă de tip **întreg** care reține valoarea 4 și **d** o variabilă de tip **real** care reține valoarea **7.3**. Efectuăm atribuirea **d←c+1**. Înainte de atribuire cele două variabile conțin:

După atribuire conținutul lor va fi:



Explicație. Am văzut faptul că este posibil ca unei variabile de tip *real* să-i atribuim o valoare întreagă. Este normal să fie așa pentru *că mulțimea numerelor întregi este submulțime a numerelor reale.*

La ce folosește atribuirea? Atribuirea are un rol uriaș în programare. Să încercăm o sistematizare a cazurilor când folosim atribuirea.

1. Inițializări.

Aproape orice program folosește variabile. *Cum facem ca acestea să conțină o valoare pe care o dorim?* O posibilitate (singura studiată până în prezent) ar fi să citim valorile respective. Procedeul este greoi și chiar caraghios în cazul în care valoarea de pornire este întotdeauna aceeași. Să ne imaginăm că se dorește ca valoarea inițială a trei variabile (**a,b,c**) să fie **0**. Ce facem? Întrebăm care este valoarea lui **a**, tastăm **0**, întrebăm care este valoarea lui **b**, tastăm **0** ș.a.m.d. Să fim serioși! Valoarea inițială poate fi stabilită prin trei instrucțiuni de atribuire, ca mai jos:

```
a ← 0;  b ← 0; c ← 0;
```

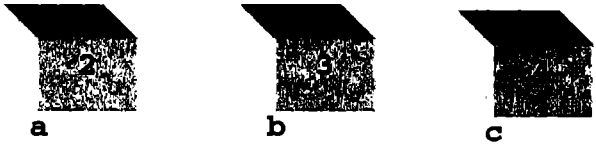
Stabilirea valorilor cu care anumite variabile intră în calcule se numește inițializare. Inițializările se fac cu ajutorul instrucțiunii de atribuire.

2. Calcule.

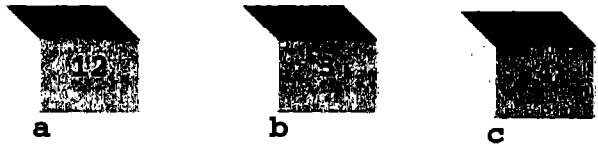
Majoritatea programelor efectuează calcule. În afara tipăririi imediate a rezultatului evaluării unei expresii, există și posibilitatea ca acesta să fie păstrat într-o variabilă (de multe ori este imposibil să procedăm altfel). În astfel de cazuri se folosește instrucțiunea de atribuire. Există, în principal, două forme în care atribuirea intervine în calcule.

Forma directă. *Unei variabile i se atribuie o expresie cu operanzi variabile, constante, alții decât variabila care va reține rezultatul. După atribuire, toți operanzii rămân nemodificați.*

Exemplu. Fie **a, b** și **c** trei variabile de tip **întreg**, care conțin, respectiv, valorile **2, 3, 4**.

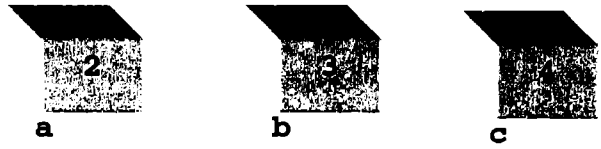


Se efectuează atribuirea **a←b*c**. În urma atribuirii, conținutul celor trei variabile devine:

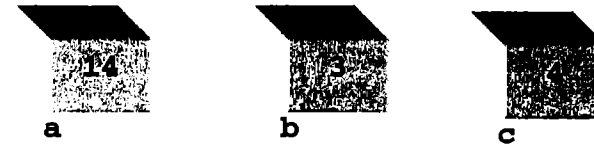


Observăm că valoarea inițială a variabilei **a** s-a pierdut. În locul ei, a fost reținut produsul valorilor reținute de variabilele ***b*** și ***c***. Datorită acestei forme de atribuire, se face deseori confuzia între atribuire și egalitate. Atenție!

Forma indirectă. *Unei variabile i se atribuie o expresie cu operanzi variabile, constante, în care intră și valoarea pe care o reține variabila căreia i se face atribuirea.* Revenim la exemplul precedent (aceleași variabile cu aceleași valori inițiale).



Efectuăm atribuirea: **a←a+b*c**. Iată ce reține cele trei variabile:



S-a calculat produsul dintre valorile reținute de **b** și **c**. Rezultatului i s-a adăugat valoarea inițială a lui **a**. Cei care confundă atribuirea cu egalitatea vor rămâne dezamăgiți. Nu vor înțelege cum nu se reduce **a** cu **a** și nu rămâne **b*c=0**... Lăsând gluma la o parte, forma indirectă de atribuire este extrem de utilizată.

3. **Copiere**. De multe ori, este necesar ca o valoare reținută de o variabilă să fie reținută și de alta. Motivele sunt diverse. O simplă instrucțiune de atribuire ne rezolvă problema de mai sus. De exemplu, dacă **x** și **y** sunt două variabile de tip *real* care rețin două valori oarecare, și dorim ca variabila **x** să rețină valoarea pe care o reține y, efectuăm atribuirea: **x←y**. Atenție! *După atribuire, conținutul variabilei y rămâne neschimbat*. Rămâne să precizăm că un raționament de genul: dacă variabila **x** a preluat valoarea pe care o reține **y**, rezultă că **y** nu mai reține nici o valoare, nu are ce căuta la programare. Ca și cum am avea două valize, și dacă luăm o cămașă din una și o punem în cealaltă, prima valiză nu mai conține cămașa mutată. Un caz particular, dar des întâlnit în programare este *interschimbarea conținutului a două variabile* (evident, de același tip).

Fie variabila de tip **întreg x** care reține valoarea **1** și variabila de tip **întreg y** care reține valoarea **2**. Dorim să inversăm conținutul celor două variabile, adică **x** să rețină **2** și **y** să rețină **1**. Se cere ca secvența să rămână valabilă indiferent ce valori ar reține **x** și **y**. Ce putem face? Dacă executăm atribuirea **y←x;** nu am rezolvat nimic întrucât continutul lui **y** se pierde. Dacă executăm atribuirea **x←y** am pierdut conținutul variabilei **x**. Dacă executăm două atribuiri și anume **x←2; y←1;** raționamentul nu este general (dacă **x** ar fi reținut **3** și **y** valoarea **4** secvența nu ar fi fost valabilă). Atunci?

Pentru rezolvare, folosim și o altă variabilă, de același tip, pe care o notăm **m**. Ea are rolul unei variabile auxiliare (nu ne folosește decât ca să realizăm interschimbul de valori). Iată pașii de lucru:

- variabila **m** va prelua valoarea reținută de **y; m←y;**
- **y** va prelua valoarea reținută de **x; y←x;**
- **x** va prelua valoarea reținută de **m; x←m.**

Reprezentăm grafic pașii folosiți. Situația inițială:

m←y;

```
y←x;
```


```
x←m;
```

Observăm că valorile reținute de **x** și **y** au fost inversate. Am fi putut folosi și secvența (de ce?):

```
m←x;  x←y;  y←m;
```

1.5.3. Operații de decizie

În pseudocod operația de decizie are forma următoare:

```
Dacă condiție atunci
|    operație1
| altfel
|  operație2
|_■
```

Modul de executare este următorul:

1. *Se testează condiția (condiția este o expresie logică);*
2. *Dacă condiția este îndeplinită (ia valoarea* **true***), se execută operația$_1$, altfel (ia valoarea* **false***) se execută operația$_2$.*
3. *În continuare se execută operația aflată după operația decizională.*

Observație. Decizia constă în a alege o operație sau alta spre a fi executată (în nici un caz amândouă). Să dăm un exemplu: *se citesc două valori întregi a și b. Se cere să se tipărească cea mai mare dintre ele.*

```
întreg a,b
Citește a
Citește b
Dacă a>b  atunci
   |    Scrie a
   | altfel
   |    Scrie b
   |_■
```

În primul rând, se testează condiția. În exemplul nostru ea este **a>b.** În cazul în care valoarea reținută de variabila **a** este mai mare decât cea reținută de variabila **b,** condiția este îndeplinită și se tipărește **a**. Contrar, se tipărește **b.**

Avem posibilitatea ca operațiile subordonate operației decizionale să fie tot operații decizionale. Exemplu: se citesc **4** valori reale **a, b, c, d.** Să se evalueze expresia:

$$E=\begin{cases} a+b, & c+d>0 \\ a-b, & c+d=0 \\ a\cdot b, & c+d<0 \end{cases}$$

Exemplu numeric. Fie **a=1, b=2, c=3, d=4**. Avem **c+d=7>0**, rezultă că se va tipări **a+b=1+2=3**. Analizați algoritmul care urmează:

```
real    a, b, c, d;
Citește a, b, c, d
 Dacă c+d>0  atunci
      |     Scrie a+b
      |   altfel
      |    Dacă c+d=0
      |      |    atunci
      |      |      Scrie a-b
      |      |    altfel
      |      |     Scrie a*b
      |      |_■
      |_■
```

Dacă suma **c+d** nu este mai mare decât **0,** rezultă că este mai mică sau egală cu **0**. Din acest motiv, clauza **altfel** a primului **dacă** va conține o altă operație decizională (**dacă**) în urma căreia se va decide dacă **c+d=0** sau **c+d<0**. În cazul în care **c+d** nu este **0**, singura posibilitate este **c+d<0** (pentru că deja cunoaștem că nu este mai mare ca **0**). Acesta este motivul pentru care nu mai facem testul **c+d<0**.

✓ Observați faptul că expresiile se pot calcula și tipări prin utilizarea operației **Scrie.**

O problemă interesantă este și aceasta: *dacă algoritmul impune ca operația care urmează lui* **altfel** *să lipsească, cum procedăm?*

Pseudocodul admite o formă particulară a operației decizionale.

```
Dacă condiție atunci
|   operație
|_■
```

Principiul de executare:

1. *Se testează condiția;*

2. *În cazul în care aceasta este îndeplinită, se execută operația aflată după* **atunci**, *altfel se trece la operația următoare.*

Exemplu. Se citește o valoare întreagă. În cazul în care aceasta este pară (se împarte exact la **2**) se va afișa mesajul **"am citit un numar par"**. Altfel, programul nu va da mesaj.

```
întreg nr;
Citește nr
Dacă ⌈nr/2⌉ = nr/2 atunci
|  Scrie 'am citit un numar par'
|_■
```

Cum putem greși?

O eroare, des întâlnită este așa numita *eroare a testului inutil.* Reluăm exemplul în care evaluăm expresia:

$$E=\begin{cases} a+b, & c+d>0 \\ a-b, & c+d=0 \\ a\cdot b, & c+d<0 \end{cases}$$

Unii scriu instrucțiunea decizională astfel:

```
Citește a, b, c, d
 Dacă c+d>0  atunci
        |     Scrie a+b
        |  altfel
        |    Dacă c+d=0 atunci
        |    |    Scrie a-b
        |    | altfel
        |    |    Dacă c+d<0  atunci
        |    |    |    Scrie a*b
        |    |    |_■
        |    |
        |    |_■
        |
        |_■
```

Dacă **c+d>0** nu este adevărat, înseamnă că este mai mic sau egal cu **0**, și dacă nu este **0**, este în mod sigur strict mai mic ca **0**. Nu are rost să facem un test suplimentar. Calculatorul nu gândește, îl face, programul funcționează corect, dar se pierde inutil timp de calcul.

Probleme propuse

1. Dorim să calculăm suma **99 + 1**. Concepem un procedeu de calcul astfel:
   - se așează **1** pe prima poziție;
   - se adaugă două cifre **0**.

   Este acesta un algoritm?

2. Există un algoritm capabil să calculeze toate zecimalele lui π?

3. Ce va reține variabila **x** după secvența de atribuiri care urmează?
   x←3; y←1; x←x+x; y←x+y; x←y.

4. Ce se afișează în urma executării secvenței, dacă se citesc, pe rând, valorile **3** și **7**?

```
întreg x
Citește x
Citește x
Scrie x
Scrie x
```

5. Elaborați algoritmul care citește pe **x** (o valoare reală) și calculează funcția de mai jos:

$$f(x)=\begin{cases} x+3, & x\le -15; \\ x^2-x, & -15<x\le 0; \\ \dfrac{1}{x}, & x>0. \end{cases}$$

6. Elaborați algoritmul care citește pe **x** (o valoare reală) și calculează **f(g(x))**.

$$f(x)=\begin{cases} x, & x\le 0; \\ x^2, & 0<x\le 10; \\ x^3, & 10<x\le 20; \\ x^4, & 20<x. \end{cases} \qquad g(x)=\begin{cases} \dfrac{1}{x}, & x<0; \\ 2, & x=0; \\ x+[x], & x>0. \end{cases}$$

7. Se citesc **4** numere reale **a**, **b**, **c** și **d** care îndeplinesc condițiile: **a<b**, **c<d**. Să se scrie un algoritm în pseudocod prin care se decide dacă intervalele **[a,b]**, **[c,d]** au sau nu intersecția nevidă.

8. Se citesc 4 numere naturale **a**, **b**, **c**, și **d**. Știind că punctele **P(a,b)** și **Q(c,d)** sunt două dintre vârfurile uniui dreptunghi cu laturile paralele cu axele **OX** și **OY**, să se scrie algoritmul pseudocod prin care se decide dacă cele două vârfuri se găsesc pe o latură a dreptunghiului sau pe o diagonală a acestuia.

9. Ce se afișează în urma executării secvenței de mai jos?

```
a←3 b:←4 c←5
a←a+b-c b←a-b+c c←c-a+b
scrie  a, b, c
```

a) 3 4 5; **b)** 5 4 3; **c)** 2 3 6; **d)** 6 5 9.

10. În condițiile în care variabila **a** reține 3, variabila **b** reține 4 și variabila **c** reține 5, care este ordinea în care trebuie să fie scrise atribuirile următoare, astfel încât, după efectuarea lor, expresia **a+b+c** să rețină valoarea maximă în raport cu ordinea atribuirilor.

```
1) a←a+b-c 2) b←a-b+c 3) c←c-a+b
```

a) nu contează ordinea;
b) 1) 2) 3);
c) 2) 1) 3);
d) 3) 1) 2).

11. În condițiile în care a reține 3 și **b** reține 5, de câte ori trebuie executată atribuirea **a←2*a-b** (prin * am notat operatorul de înmulțire), astfel încât **a** să rețină **-27**.

a) O dată; **b)** De două ori; **c)** De trei ori; **d)** De patru ori.

12. În condițiile în care **a** reține 3 și **b** reține 9, de câte ori trebuie executată secvența următoare, asfel încât valoarea reținută de **a** să fie egală cu valoarea reținută de **b**.

```
a←a+1; b←b-1.
```

a) O dată; **b)** De două ori; **c)** De trei ori; **d)** De patru ori.

13. În condițiile în care **a** reține 3 și **b** reține 6, de câte ori trebuie executată secvența următoare, asfel încât valoarea reținută de **a** să fie egală cu valoarea reținută de **b**.

```
a←a+1; b←b-1.
```

a) Indiferent de câte ori se execută secvența, nu vom avea egalitate;
b) O dată;
c) De două ori;
d) De trei ori.

14. Fie **a** și **b** două variabile de tip întreg cu un conținut neprecizat. Dacă se execută atribuirile următoare, care dintre relațiile următoare nu este adevărată?

```
a←b-1; b←a+1
```

a) a+b=2b-1; **b)** |a-b|=1; **c)** a-b=1; **d)** b-a=1.

15. Ce valoare se va afișa dacă se citește **7**? Prin * am notat operatorul de înmulțire, iar prin **div** operatorul de împărțire întreagă. De exemplu, dacă **x =5**, **x div 2** produce valoarea **2**.

```
întreg x
Citește x
Dacă x>5  atunci
   |   x←x div 2
   | altfel
   |   x:=x*x
   |_■
Dacă x>5  atunci
   |   x←x div 2
   | altfel
   |   x:=x*x
   |_■

Scrie x;
```

a) 2; **b)** 25; **c)** 12; **d)** 9.

Testele **16** și **17** se referă la pseudocodul de mai jos:

```
întreg x
Citește x
Dacă x>5  atunci
   |   x←x+1
   | altfel
   |   Dacă x≤3  atunci
   |   |   x←x+2
   |   | altfel
   |   |   x:=x+3
   |   |_■
   |
   |_■
Scrie x;
```

16. Ce se va afișa dacă **x=4**?

a) 5; **b)** 6; **c)** 7; **d)** 8.

17. Care este intervalul căruia trebuie să-i aparțină **x**, astfel încât să se afișeze **x+3**?

a) $(-\infty,3)$; **b)** $(-\infty,3]$; **c)** $(3,5]$; **d)** $[3,5)$.

18. Pentru **x** și **y** citite, algoritmul următor afișează **1**. Care dintre afirmațiile de mai jos este adevărată?

```
Întreg x,y
Citește x
Citește y
Dacă x+y>0 atunci
   |   Scrie 1
   |_■
```

a) Au fost citite două numere pozitive;
b) Au fost citite două numere de semn contrar;
c) Au fost citite două numere nenule;
d) Cel puțin una dintre valorile citite este pozitivă.

Testele de la **19** la **21** se referă la pseudocodul de mai jos:

```
Întreg x,f
Citește x
Dacă x<3  atunci
   |   f←x+1
   | altfel
   |   Dacă x>10  atunci
   |   |   f←x+2
   |   | altfel
   |   |     Dacă x>0  atunci
   |   |       | f←x
   |   |       |  altfel
   |   |       |   f:=0
   |   |       |_■
   |   |_■
   |
   |_■
Scrie f;
```

19. Care este numărul valorilor de intrare pentru care se afișează **0**?

a) 0; **b)** 2; **c)** 1; **d)** o infinitate.

20. Care este numărul valorilor de intrare pentru care se afișează **x**?

a) 0; **b)** 7; **c)** 8; **d)** o infinitate.

21. Rescrieți algoritmul astfel încât la aceleași intrări să fie aceleași ieșiri (veți obține un algoritm echivalent), dar, veți utiliza o singură variabilă și două operații decizionale.

22. Fie **a<b** două numere reale. Care secvență testează dacă **x**, o valoare reală aparține intervalului **[a,b)**?

a) Dacă (x≥a) SAU (x<b) atunci...
b) Dacă (x≥a) SI (x<b) atunci...
c) Dacă (x-a)*(b-x)≥0 atunci...
d) Dacă (x-a)*(x-b)<0 atunci...

23. Care dintre expresiile logice de mai jos este echivalentă cu expresia:

NOT((x≤a) SI (x>b))?

a) NOT(x≤a) SI NOT(x>b);
b) (x>a) SI (x≥b);
c) (x>a) SAU (x≤b);
d) (x<a) SAU (x>b).

24. Dacă **a=3, b=4, c=5**, care dintre expresiile de mai jos produce valoarea **true**?

a) (a-b)*(b-c)*(c-a)>0;
b) (a-b)+(b-c)+(c-a)>0;
c) (a-b)-(b-c)-(c-a)>0;
d) (a>b) SAU (b>c) SAU (a>c).

25. Care dintre secvențele de mai jos nu este echivalentă cu celelalte 3? Variabila **a** are tipul logic, iar variabila **b** este de tip real.

a)
```
a←b>5
Dacă a atunci
|  Scrie 1
|_■
```

b)
```
a←NOT (b≤5)
Dacă a atunci
|  Scrie 1
|_■
```

c)
```
a←(b>5) SI (b>4)
Dacă a atunci
|  Scrie 1
|_■
```

d)
```
a←(b>5) SI (b>6)
Dacă a atunci
|  Scrie 1
|_■
```

26. Dacă **a** este o variabilă reală, care dintre secvențele de mai jos nu poate fi înlocuită cu: **scrie a** ?

a)
```
Dacă (a>1) SAU (a≤1)
| atunci
| Scrie a
|_■
```

b)
```
Dacă a²-a+1>0
| atunci
| Scrie a
|_■
```

c)
```
Dacă a≤a²
| atunci
| Scrie a
|_■
```

d)
```
Dacă (a=0) SAU NOT(a=0)
| atunci
| Scrie a
|_■
```

27. Dacă **a** este o variabilă reală, în care dintre secvențele de mai jos există posibilitatea să se afișeze **a** ?

a)
```
Dacă (a>1) SI (a≤1)
| atunci
| Scrie a
|_■
```

b)
```
Dacă a<a³
| atunci
| Scrie a
|_■
```

c)
```
Dacă a²+a+1<0
| atunci
| Scrie a
|_■
```

d)
```
Dacă (a=0) SI NOT(a=0)
| atunci
| Scrie a
|_■
```

Răspunsuri:

9. c) În pseudocod nu este obligatoriu să se declare tipul variabilelor. Acesta se poate deduce în funcție de valorile pe care acestea le iau. **10. c) 11. d) 12. c) 13. a) 14. c) 15. d) 16. c) 17. c) 18. d) 19. a) 20. c) 22. b) 23. c) 24. a) 25. d)** Dacă **b=5.5... 26. c) 27. b)**

CAPITOLUL 2

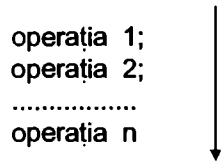
Principiile programării structurate

2.1. Introducere

Am învățat că algoritmii efectuează trei tipuri de operații:

- de intrare / ieșire;
- de atribuire;
- de decizie.

Algoritmul efectuează aceste operații în scopul transformării datelor de intrare în date de ieșire. Dar care este ordinea în care se efectuează? În mod simplificat, de la prima operație către ultima. În acest mod un algoritm arată ca mai jos, iar ordinea de executare a operațiilor este cea indicată de săgeată:



Dar ce ne facem în situația în care o operație trebuie executată de mai multe ori. *Exemplu: presupunând că nu cunoașteți formula, se cere să se calculeze suma primelor* **1000** *numere naturale nenule.* Din cele de mai sus, rezultă că ar trebui să scriem un "algoritm" de forma:

```
întreg n, s
s ← 0
s ← s+1
s ← s+2                    de 1000 de ori!
.....
s ←s+1000
Scrie s
```

Cam mult de scris, chiar și pentru cei "constiinciosi"! Lăsând gluma la o parte, e clar că trebuia găsit ceva prin care să se evite scrierea repetată.

Atunci s-a găsit o soluție, care a fost aplicată mulți ani. A fost "inventată" operația de salt numită **GO TO.** În ce constă ea? Rândurile (liniile) care conțin algoritmul pot fi etichetate printr-un nume. Operația de salt **GO TO** are forma **GO TO nume**. Aceasta înseamnă că, la întâlnirea ei, controlul executării se transferă la operația care se găsește pe linia etichetată cu nume. Apoi, dacă nu mai există o altă operație **GO TO,** se execută operația următoare celei etichetate etc. Iată cum arată în noile condiții algoritmul prin care se calculează suma primelor **1000** de numere naturale nenule:

```
întreg s, i;
    s ← 0;
    i ← 1;

iar s ← s+i;
    i ← i+1;
    Dacă i <= 1000  atunci
      |GO TO iar
      |■
    Scrie s
```

Variabila **s** este inițializată cu **0**, iar variabila **i** este inițializată cu **1**. La variabila **s** se adună conținutul variabilei **i**, adică **1**. Apoi **i** ia valoarea **2**. Întrucât conținutul variabilei **i** este mai mai mic decât **1000**, controlul se transferă la operația aflată pe linia etichetată cu **iar**. Aceasta înseamnă că se adună **2** la **s**, deci conținutul acesteia devine **3**. Procedeul se reia până când **i** devine mai mare decât **1000**. În final se afișează **s**, adică suma primelor **1000** numere naturale nenule. În acest fel *am programat un ciclu, adică o secvență de operații care se repetă*. Cam așa se redactau algoritmii...

În timp, s-au conturat anumite dezavantaje ale unui astfel de mod de programare. În primul rând, odată scris, după o vreme, *algoritmul devenea greu de înțeles chiar și pentru cel care l-a elaborat*. Acesta putea să conțină mai multe cicluri unul în altul (se numesc imbricate). Erau, deci, necesare mai multe operații de tip **GO TO**. Apoi, au apărut dificultăți chiar în etapa de elaborare a algoritmilor, *pentru că în loc să ne concentrăm asupra operațiilor esențiale, tot testam cum am pus etichetele pentru* **GO TO** *etc.*

Programarea structurată a apărut la începutul anilor 70. Este un concept cu o importanță fundamentală în scrierea algoritmilor. Cine învață să redacteze structurat algoritmii dispune de mari avantaje, cum ar fi:

- o mai mare ușurință de scriere a algoritmilor;
- odată scris, un algoritm se înțelege mult mai ușor (înainte de apariția programării structurate, după o perioadă, nici cel care redactase algoritmul nu îl mai înțelegea).

În general, algoritmii se elaborează prin utilizarea *exclusivă* a anumitor structuri (liniară, alternativă, repetitivă) care vor fi prezentate în acest capitol.

Prin structură înțelegem o anumită formă de îmbinare a operațiilor cu care lucrează algoritmii.

Desigur, pentru a fi siguri că acele structuri sunt suficiente pentru elaborarea *oricărui* algoritm au fost necesare anumite demonstrații. Acestea au fost făcute și nu constituie obiectul actualului curs.

Doresc să lămuresc o anumită problemă. Programarea structurată nu înseamnă doar eliminarea operației de salt **GO TO**. Înseamnă cu mult mai mult, *un mod de gândire a algoritmilor*. Acest mod rezultă din gândirea acestora doar prin structurile permise. *Problema pentru care trebuie să elaborăm algoritmul se descompune, după structurile amintite, în subprobleme. Apoi acestea se descompun, din nou, după aceleași reguli, până se ajunge doar la operații elementare.*

În acest manual vom învăța să redactăm algoritmii direct structurat, varianta nestructurată ținând deja de istoria informaticii.

2.2. Structuri de bază, descrierea acestora în pseudocod

Structurile de bază utilizate în descrierea algoritmilor sunt de trei feluri:

- structura liniară;
- structura alternativă;
- structura repetitivă.

2.2.1. Structura liniară

Vom defini structura liniară astfel:

➢ Orice operație (citire / scriere, atribuire, decizională considerată în ansamblul ei) constituie o structură liniară;

➢ Dacă **S1** și **S2** sunt structuri (de orice tip), atunci:

S1
S2

este structură liniară.

Pornind de la definiție, se ajunge la următoarea formă de structură liniară:

operația 1
operația 2
.....
operația n.

De ce? Operația **1** este structură liniară, operația **2** este structură liniară, rezultă că operația **1**, urmată de operația **2** este structură liniară și cum operația **3** este structură liniară, înseamnă că operația **1** urmată de operația **2**, urmată de operația **3** este structură liniară ș.a.m.d. Ordinea de execuțare este: operația **1**, apoi operația **2**, ..., până la operația **n.**

Exemplul 1. *Se citesc două variabile reale. Să se interschimbe conținutul lor și să se afișeze.*

```
real a, b, man;
Citește a, b
man ← a;
a ←b;
b ← man;
Scrie a, b
```

- se citesc cele două variabile - operația 1, de citire;
- variabilei **man** i se atribuie conținutul variabilei **a** - operația 2, de atribuire;
- variabilei **a** i se atribuie conținutul variabilei **b** - operația 3, de atribuire;
- variabilei **b** i se atribuie conținutul variabilei **man** - operația 4, de atribuire;
- se afișează **a, b** - operația 5.

Exemplul 2. *Se citesc două valori reale. Să se afișeze valoarea cea mai mică.*

```
real a, b;
Citește a, b
Dacă a < b atunci scrie a
          altfel scrie b
```

✓ Am utilizat o formă simplificată a operației decizionale!

Prima operație este de citire. A doua operație este cea decizională - aceasta privită în ansamblul ei. Ea subordonează alte două operații de tipărire (scriere).

Exemplul 3. *Se citesc două variabile întregi. Să se interschimbe conținutul lor, fără a folosi o variabilă auxiliară de manevră.*

```
întreg a, b;
Citește a, b
a ←a+b
b ← a-b
a ← a-b
Scrie a, b
```

Pentru valorile citite ale lui **a** și **b** (**a=3, b=4**) algoritmul funcționează astfel:

- lui **a** i se atribuie valoarea **3+4=7**;
- lui **b** i se atribuie valoarea **7-4=3**;
- lui **a** i se atribuie valoarea **7-3=4**;
- se afișează **a=4** și **b=3** (rezultat corect).

Exemplul 4. *Se citesc două valori întregi* **a** *și* **b**. *Se cere să se afișeze media lor aritmetică.*

Problema are o capcană. Din faptul că valorile citite sunt întregi, nu trebuie trasă concluzia că media aritmetică este număr întreg (de exemplu dacă citim **8** și **9** media va fi **8,50**). Aceasta înseamnă că variabila care reține media trebuie să fie de tip **real**.

```
întreg a, b;
real medie;
Citește a, b
medie ← (a+b)/2
Scrie medie
```

Exemplul 5. *Se citește un număr natural format din 3 cifre. Se cere să se afișeze suma cifrelor sale. Exemplu: dacă citim 123 se va afișa 6.*

Cum rezolvăm? Dacă o variabilă reține un număr, nu putem în mod direct să vedem care sunt cifrele sale. Trebuie să găsim ceva care să ne permită accesul la ele. Cunoaștem faptul că restul împărțirii la **10** al unui număr este ultima sa cifră. Mulți vor fi tentați să găsească prima cifră a numărului și nu ultima. Dar ce importanță are în ce ordine adunăm cifrele? Putem aduna **1+2+3** dar și **3+2+1**. Rezultatul este același. Tot este bine că putem obține ultima cifră. Ce facem cu celelalte? Dacă am reuși să obținem numărul fără ultima cifră problema ar fi rezolvată (am putea repeta procedeul). Este posibil. Numărul fără ultima cifră este dat de câtul întreg dintre număr și **10**. În concluzie, vom obține suma cifrelor numărului citit adunând, de fiecare dată, ultima cifră și obținând numărul fără ea.

```
întreg x, s;
Citește x
s ← 0
s ← s+x mod 10
x ← x div 10
s ← s+x mod 10
x ← x div 10
s ← s +x mod 10
Scrie s
```

- ✓ Operația prin care obținem restul împărțirii unui număr la o valoare am notat-o cu **mod**, iar cea prin care obținem câtul întreg al unei împărțiri am notat-o cu **div**. Exemple:

123 mod 10 =3

123 div 10=12.

- ✓ În pseudocod putem simboliza operațiile așa cum dorim. Condiția este să ținem minte ce am notat prin operația respectivă (eventual să precizăm aceasta alăturat, printr-un mic comentariu).
- ✓ Observați faptul că la variabila **s**, care a fost inițializată cu **0**, s-a adunat de fiecare dată ultima cifră. Dar nu vă deranjează faptul că

aceeași secvență a fost repetată de **3** ori? Dar dacă numărul avea mai multe cifre? Vom învăța că astfel de algoritmi pot fi scriși mult mai eficient, prin utilizarea structurilor repetitive.

Exemplul 6. *Se citesc 3 numere naturale. Se cere să se afișeze primul număr, suma dintre primul și al doilea, suma primelor 3 numere. Exemplu: dacă se citește 2, 5 și 7, se va afișa 2, 7, 14.*

Pentru rezolvare, se adună la o variabilă **s** inițializată cu **0**, pe rând, cele trei numere. Ce observăm? Faptul că se citesc **3** numere nu înseamnă că trebuie să rezervăm **3** variabile pentru citire. Este suficient să avem una (**nr**). La prima citire ea va reține primul număr. După ce acesta a fost adunat la conținutul variabilei **s**, nu mai avem nevoie ca programul să-l rețină. Citirea în același loc (aceeași variabilă) a numărului următor (se citește tot variabila **nr**) conduce la pierderea conținutului inițial al variabilei. Și aici, deranjează faptul că o secvență este repetată.

```
întreg s, nr;
s ← 0
Citește nr    s ← s+nr  Scrie s
Citește nr    s ← s+nr  Scrie s
Citește nr    s ← s+nr  Scrie s
```

2.2.2. Structura alternativă

Structura alternativă se definește astfel:

- Dacă S1 și S2 sunt structuri și E este o condiție, atunci:

```
Dacă E atunci
  |      S1
  |  altfel
  |      S2
  |_■
```

este structură alternativă.

Mecanismul de executare este:

- se evaluează condiția;
- dacă aceasta este îndeplinită se execută **S1**, altfel se execută **S2**.

Forma simplificată a structurii alternative este:

- Dacă S este o structură și E este o condiție, atunci:

```
Dacă E atunci
  |      S1
  |_■
```

este structură alternativă.

Dacă la evaluare **E** este îndeplinită se execută **S**, altfel se trece mai departe.

Exemplul 1. *Să se scrie un algoritm care citește un număr natural (comanda). Dacă acesta este 0 se vor citi două numere întregi a și b și se va afișa suma lor, contrar se vor citi două numere reale x și y și se va afișa suma lor.*

În ambele cazuri trebuie executate mai multe instrucțiuni. Pentru aceasta am folosit structura alternativă.

```
întreg comanda, a, b, s1;
real  x, y, s2;
Citește comanda
Dacă comanda = 0  atunci
     |      Citește a,b
     |      s1 ←a+b
     |      Scrie s1
     |    altfel
     |     Citește x, y
     |     s2 ←x+y
     |     Scrie s2
     |■
```

```
Citește a,b          Citește x,y
s1 ←a+b              s2 ←a+b
Scrie s1             Scrie s2
    ↑                    ↑
   P1                   P2
```

Cele două secvențe de mai sus sunt structuri liniare. Le vom nota cu P_1 și P_2.

În aceste condiții algoritmul în pseudocod devine:

```
întreg comanda, a, b, s1;
real  x, y, s2;
Citește comanda
Dacă comanda = 0  atunci
     |      P1
     |    altfel
     |      P2
     |■
```

Structura alternativă, în asamblul ei, o vom nota cu **A**. În aceste condiții algoritmul devine:

```
întreg comanda, a, b, s1
real  x, y, s2;
Citește comanda
A
```

Cele două structuri (facem abstracție de declarații) alcătuiesc o structură liniară, pe care o notăm cu **P**.

În aceste condiții "rezolvarea" este:

```
Declarații de variabile

P
```

După cum observați, totul s-a redus la o singură structură prin respectarea regulilor impuse de programarea structurată.

În realitate, dvs. veți proceda invers atunci când scrieți un algoritm: prin respectarea regulilor programării structurate, descompuneți problema în subprobleme, pe acestea le descompuneți în mod asemănător, până când ați obținut doar operații elementare.

Exemplul 2. Se citesc **a** și **b**, numere reale. Să se rezolve ecuația de gradul 1: $ax + b = 0$.

În matematică, ecuația de gradul **1** este definită numai dacă **a≠0**. În informatică, trebuie tratat și cazul **a=0**, pentru că nimeni nu garantează că utilizatorul nu greșește la introducerea datelor.

Cum gândim? Într-o primă etapă algoritmul nostru va testa dacă **a** este diferit de **0**. În caz afirmativ, se va rezolva ecuația în mod clasic, contrar se va trata excepția. Atât **Rezolvă ecuația** cât și **Tratează excepția** sunt structuri pe care le vom trata în etapa următoare.

```
real a,b,x
Citește a,b
Dacă a≠0 atunci
  |   Rezolvă ecuația
  | altfel
  |   Tratează excepția
  |■
```

Rezolvarea ecuației presupune calculul și afișarea rădăcinii (ambele sunt operații elementare), iar dacă **a=0**, rămâne să vedem cum este **b** pentru a determina dacă ecuația nu admite soluții (**b≠0**), sau admite o infinitate de soluții (**b=0**).

Iată algoritmul:

```
real a,b,x
Citeşte a,b
Dacă a≠0 atunci
  |     x ← -b/a
  |     Scrie x
  | altfel
  |   Dacă b≠0 atunci
  |     |     Scrie " Ecuaţia nu admite soluţii"
  |     | altfel
  |     |     Scrie " Ecuaţia admite o infinitate de soluţii"
  |     |■
  |■
```

✓ În vreme ce operaţia decizională subordonează sub **dacă** sau sub **altfel** câte o singură operaţie, structura alternativă subordonează sub **dacă** sau sub **altfel** câte o singură structură! Cu alte cuvinte, structura alternativă generalizează operaţia decizională.

2.2.3. Structura repetitivă

Structura repetitivă apare în următoarele variante:

- condiţionată anterior - este de două feluri:
  - **Cât timp... execută (While Do)**
  - **Pentru... execută (For)**
- condiţionată posterior:
  - **Repetă ... până când (Repeat until)**
  - **Execută ... cât timp (Do While);**

Singura structură indispensabilă este **Cât timp... execută (While Do)** - restul se poate obţine din aceasta.

2.2.3.1. Structura Cât timp... execută (While Do)

Structura de tip **Cât timp... execută** se defineşte astfel:

➢ Dacă **E** este condiţie şi **S** o structură atunci

```
Cât timp E execută
  | S
  |■
```

este o structură de tip **Cât timp ... execută**

Principiul de executare este următorul:

P1 Se evaluează condiția. Dacă este îndeplinită, se execută **S** și se trece la P2, altfel executarea se încheie;
P2. Se reia pasul **P1**.

Observație. Se observă faptul că **S** se execută numai dacă condiția este îndeplinită. Din acest motiv, structura se mai numește condiționată anterior. De la bun început precizăm următoarele:

⇒ este obligatoriu să folosim structura **Cât timp... execută** atunci când sunt îndeplinite simultan două condiții:

- instrucțiunea subordonată se execută (de câte ori este cazul) numai dacă este îndeplinită o anumită condiție;
- în momentul în care se intră în secvența repetitivă nu se știe de câte ori aceasta se va executa.

Exemplul 1. *Se citește un număr natural* **n**, *diferit de* **0**. *Să se afișeze suma cifrelor sale.* Dacă se citește **248** se va afișa **14**.

```
întreg i, n, s;
Citește n
s ← 0
Cât timp  n <> 0 execută
  |  s  ←  s+n mod 10;
  |  n  ←  n div 10
  |■
Scrie s
```

Exemplu numeric. Se citește **n = 248.**

248≠0, 248 mod 10 = 8, s=0+8=8; n=24;
24≠0, 24 mod 10 = 4, s=8+4=12; n=2;
2≠0, 2 div 10 = 0, s=12+2=14; n=0;
0=0, se afișează **s=14.**

Exemplul 2. *Se citește un număr natural* **n**. *Să se afișeze numărul obținut prin inversarea numărului citit.* Exemplu: Dacă citim **n=248** se va afișa **842**.

Să considerăm două numere: al doilea număr a fost obținut din primul prin adăugarea la sfârșit a unei cifre (**456** și **4563**). Avem relația: **4563=456×10+3**. Evident, relația este adevărată pentru oricare două numere naturale care îndeplinesc condiția de mai sus. Este clar că numărul inversat va începe cu ultima cifră a numărului citit. Să presupunem că am citit **248**. Vom construi numărul inversat astfel:

8=0×10+8, (**8** este ultima cifră a numărului citit și rămâne **24**).
84=8×10+4, (**4** este ultima cifră a numărului rămas și se obține **2**)
842=84×10+2. (**2** este ultima cifră a numărului **2** și se obține **0**).

```
întreg i, n, s;
Citește n
s ← 0
Cât timp  n <> 0 execută
    |  s ←  s*10+n mod 10
    |  n ← n div 10
    |■
Scrie s
```

Exemplul 3. *Se citesc numerele naturale* **n1** *și* **n2**. *Să se calculeze produsul lor, fără a utiliza operatorul de înmulțire.*

Algoritmul se bazează pe faptul că înmulțirea este o adunare repetată. De exemplu, dacă citim **3** și **4**, calculăm produsul astfel **3+3+3+3=12** (am adunat de **4** ori). Adunarea repetată se face utilizând structura **Cât timp ...execută**.

Poate că vă veți întreba de ce am folosit **Cât timp n <> 0 execută** (atât timp cât, este evident, numărul de executări ale instrucțiunii subordonate lui **Cât timp... execută** este cunoscut de la bun început). Așa este. Dar dacă folosirea structurii **Cât timp ...execută** nu este indispensabilă, nu înseamnă că nu avem voie s-o folosim.

```
întreg n1, n2, s, i;
Citește n1, n2
s  ← 0
i  ← 1
Cât timp i <= n2 execută
      | s ← s+n1
      | i ← i+1
      |■
Scrie  s
```

Exemplul 4. *Se citesc două numere naturale* **n1** *și* **n2**. *Se cere să se calculeze câtul și restul împărțirii întregi a lui* **n1** *la* **n2**, *fără a utiliza operatori de împărțire.*

Cum procedăm? Așa cum înmulțirea se poate face prin adunare repetată, tot așa împărțirea se poate face prin scădere repetată. Din deîmpărțit (**n1**) se scade (dacă este posibil - dacă deîmpărțitul este mai mare sau egal cu împărțitorul) împărțitorul (**n2**), și la fiecare scădere se adună **1** la cât (inițial **0**). Când ne oprim? Atunci când valoarea rămasă (în urma scăderilor repetate din deîmpărțit a împărțitorului) este mai mică decât împărțitorul.

Exemplu numeric.: **n1=10, n2=3.**

10-3≥0; n1=10-3=7; c=1;
7-3≥0; n1=7-3=4; c=2;
4-3≥0; n1=4-3=1; c=3;
1-3<0; se iese din ciclu se afișează câtul (**3**) și restul **1**.

```
întreg n1, n2, c;
Citește n1, n2
c ← 0
Cât timp  n1-n2 >= 0 execută
    |  c  ← c+1;
    |  n1 ← n1-n2
    |■
Scrie c, n1
```

2.2.3.2. Structura de tip **Pentru...execută**

➢ Fie **i** o variabilă de tip **întreg**, numită variabilă de ciclare, **a** și **b** două valori întregi (sau două variabile de tip **întreg**) numite *valoare inițială* și *valoare finală*, și **S** o structură. Atunci:

```
Pentru i ←a,b execută
  | S
  |■
```

este o structură de tip **Pentru ...execută**

Principiul de executare este următorul:

P1. Variabila de ciclare **i** ia valoarea inițială **a**;
P2. Dacă **i** este mai mic sau egal cu **b**, se execută **S**, se adună **1** la **i** și se reia **P2**. Altfel, executarea este încheiată.

Această structură poate fi simulată cu **Cât timp... execută** astfel:

```
i ← a;
Cât timp i <= b execută
    |  S
    |  i ← i+1
    |■
```

Se utilizează structura **Pentru...execută** dacă:

- secvența se repetă;
- se cunoaște, înaintea intrării în secvență, de câte ori trebuie să fie executată secvența.

Exemplul 1. *Se citește* **n** *(număr natural). Se cere să se efectueze suma primelor* **n** *numere naturale. Exemplu:* **n=3. S=1+2+3=6.**

Cum procedăm? Este posibil să cunoașteți formula care dă suma primelor **n** numere naturale nenule: $S_n = \frac{n \cdot (n+1)}{2}$

Dacă aplicăm această formulă problema devine foarte ușoară. Întrucât exemplul este dat pentru a învăța structura **Pentru...execută**, presupunem că nu o cunoaștem. Vom aduna, pe rând, numerele **1, 2,...,n** la conținutul unei variabile **s**. Inițial, **s** reține valoarea **0**. Cum gândim? Din start, avem de adunat **n** numere. Deci, vom avea **n** pași (la fiecare pas se adună un număr). Să notăm cu **i** numărul care se adună la **s.** Instrucțiunea de atribuire va fi: **s ← s+i.** Prima dată **i** va fi **1**. A doua oară **i** va fi **2** ș.a.m.d.
Avem:

```
i←1, s ← s+1; s=0+1=1;
i←2, s ← s+2; s=1+2=3;
i←3, s ← s+3; s=3+3=6;
i←4, s←s+4; s=6+4=10.
...
```

Prezentăm algoritmul obținut:

```
întreg n, i, s;
read n
s ← 0
Pentru i ← 1, n execută
    | s ← s+i
    |■
Scrie s
```

Exemplul 2. *Să se calculeze suma:* **S = 0,1 + 0,2 +...+ 0,9.**

Un elev bun la matematică își dă seama imediat cum se rezolvă problema cu ajutorul formulelor, sau folosind algoritmul precedent. În ce ne privește am ales varianta de mai jos. Avem **9** pași. La fiecare pas se adună un număr. Rezultatul este de tip **real** (variabila **s** are acest tip). Este adevărat, variabila de ciclare nu poate lua decât valori întregi. Recurgem la următorul artificiu: vom considera o variabilă **i** care ia valori de la **1** la **9**. La fiecare pas se adună valoarea reală **i/10**. Când **i** este **1**, se adună **1/10=0,1**, când **i** este **2** se adună **2/10=0,2** ș.a.m.d.

```
întreg i
real s
s ← 0
Pentru i ← 1, 9 execută
    | s ← s+i/10
    |■
Scrie s
```

Exemplul 3. *Se citește n număr natural. Să se calculeze suma:*

$$S = 1 + 1 \cdot 2 + 1 \cdot 2 \cdot 3 + ... + 1 \cdot 2 \cdot ... \cdot n$$

Ce avem de făcut? Observăm că avem **n** pași. La pasul **i** se adună valorea **1*2...*i**, iar **i** este cuprins între **1** și **n**. Avem:

Pasul 1. Se adună **1**;
Pasul 2. Se adună **1*2**;
...
Pasul n. Se adună **1*2*3*...*n**.

Dacă la problemele anterioare valoarea care se adună se obține ușor, aici este puțin mai complicat. De ce? Pentru că și ea trebuie calculată. Cum o calculăm? Folosim un nou ciclu? În nici un caz. Se poate mai simplu. Observăm faptul că singura diferență între valorile care se adună la pasul **i** și pasul **i+1** este că valoarea adunată la pasul **i+1** se obține înmulțind cu **i+1** valoarea adunată la pasul **i**. Exemplu: La pasul **2** se adună **1*2**. La pasul **3** se adună **1*2*3**, adică **(1*2)*3**. Să notăm cu **p** valoarea care se adună la fiecare pas. Inițial **p** este **1**. La pasul **i+1**, înainte de a fi adunată, valoarea **p** se înmulțește cu **i+1**.

```
întreg i, s, p, n;
Citește n
s ← 0
p ← 1
Pentru i ← 1, n execută
    |  p ← p*i
    |  s ← s+p
    |■
Scrie  s
```

Inițial **s = 0; p = 1**.
Pasul 1. **p←p*1; p=1*1=1; s←s+p; s=0+1=1**;
Pasul 2. **p←p*2; p=1*2=2; s←s+p; s=1+2=3**;
Pasul 3. **p←p*3; p=2*3=6; s←s+p; s=3+6=9;**
...

Exemplul 4. *Se citesc* **n** *numere întregi. Se cere să se afișeze cel mai mare număr citit. Exemplu: Avem* **n=3**, *iar numerele sunt* **-7, 9, 2**. *Se va afișa* **9**.

Pe exemplul anterior identificăm cel mai mare număr din șir astfel:

⇒ cel mai mare număr este **-7**;
⇒ **9>-7**, deci cel mai mare număr este **9**;
⇒ **2<9**, deci cel mai mare număr rămâne **9**;
⇒ **3<9**, cel mai mare număr rămâne **9**.

Am parcurs integral șirul și cel mai mare număr rămâne **9**. Vă dați seama că raționamentul de mai sus rămâne valabil pentru un șir de **n** numere (**n**, număr natural oarecare). Observăm că nu este necesar să reținem, la un anumit moment, decât un singur număr (cel analizat) precum și maximul dintre cele anterior analizate. Se citește primul număr în variabila **max**. Apoi avem **n-1** pași, pe care îi numerotăm de la **2** la **n**. La fiecare pas citim un număr în variabila **nr**, apoi îl comparăm cu valoarea reținută în variabila **max**, reținând permanent în **max** valoarea cea mai mare.

```
întreg i, max, n, nr
Citește n, max
Pentru i ← 2, n execută
    | Citește nr
    | Dacă nr > max atunci
    |     | max ← nr
    |     |■
    |■
Scrie  max
```

2.2.3.3. Structura de tip Repetă...până când

➢ Fie S o structură și E o condiție. Atunci:

```
Repetă
| S
|■
până când E
```

este o structură **Repetă...până când**

Principiul de executare este următorul:

P1 Se execută **S**;

P2 Dacă **E** este nu este îndeplinită, se trece la P1, altfel executarea este încheiată.

Observați faptul că **S** se execută întotdeauna o dată. Apoi se testează dacă se mai execută sau nu. Și această structură poate fi simulată cu ajutorul structurii **Cât timp ... execută**

```
S;
Cât timp not E execută
 | S
 |■
```

Iată cum se poate calcula suma primelor **n** (**n** ≥ 1) numere naturale prin utilizarea acestei structuri:

```
întreg n, i, s;
Citește n
i ← 1
s ← 0
Repetă
| s ← s+i
| i ← i+1
|■
până când i>n
Scrie s
```

Să vedem cum decurge algoritmul după citirea lui **n** (citim **3**):

- **i** ia valoarea **1**, iar **s** ia valoarea **0**;
- **s** ia valoarea **0+1=1**, iar **i** ia valoarea **2**;
- **i** nu este mai mare decât **3**, deci **s** ia valoarea **1+2=3**, iar **i** va lua valoarea **3**;
- **i** nu este mai mare ca **3**, deci **s** va lua valoarea **3+3=6**, iar **i** va lua valoarea **4**;
- **i** este mai mare decât **n**, deci se trece la instrucțiunea următoare, unde se afișează valoarea **6**.

✓ O astfel de structură se utilizează în limbajul **Pascal**.

2.2.3.4 Structura Repetă ... cât timp

➢ Fie S o structură și E o condiție. Atunci:

```
Repetă
| S
|■
cât timp E
```

este o structură **Repetă...cât timp.**

Această structură are același mecanism de executare ca **Repetă...până când**, diferența fiind dată de faptul că secvența **S** se execută din nou dacă **E** este îndeplinită.

✓ O astfel de structură se utilizează în limbajul **C++**.

2.3. Aplicații

Problema 1. *Se citesc trei numere întregi* **a, b, c**. *Să se afișeze în ordine crescătoare.*

Ideea de lucru este următoarea:

- se compară primele două (**a** și **b**);
- după ce se cunoaște ordinea între acestea, se încearcă plasarea lui **c** față de ele.

Dacă **a** este mai mic decât **b** trebuie văzut unde îl plasăm pe **c**. El poate fi mai mic decât **a**, deci ordinea ar fi **c, a, b** sau mai mare sau egal cu **a**. În acest din urmă caz, **c** trebuie comparat cu **b**. Dacă el este mai mic decât **b**, ordinea este **a, c, b**, în caz contrar, este **a, b, c**. Tot așa se raționează în situația în care **a** nu este mai mic decât **b** (este mai mare sau egal). În pseudocod, algoritmul arată astfel:

```
întreg a, b, c
Citește a, b, c
Dacă a < b atunci
   |    Dacă c < a atunci Scrie c, a, b
   |       |  altfel
   |       |    Dacă c < b atunci Scrie a, c, b
   |       |    |            altfel Scrie a, b, c
   |       |    |■
   |       |■
   |  altfel
   |    Dacă c < b atunci Scrie c, b, a
   |       |   altfel
   |       |    Dacă c < a atunci  Scrie b, c, a
   |       |      |            altfel  Scrie b, a, c
   |       |      |■
   |       |■
   |■
```

O altă modalitate (mai generală) de rezolvare a acestei probleme va fi descrisă în continuare. Considerăm trei numere oarecare, spre exemplu **7, 4, 1.** Avem dreptul de a inversa pozițiile a două numere alăturate. Dorim ca, în final, numerele să fie scrise în ordine crescătoare. Iată cum procedăm:

- **7** este mai mare decât **4**, se inversează cele două numere, obținând **4 7 1**;
- **7** este mai mare decât **1**, se inversează cele două numere **4 1 7**;
- **4** este mai mare decât **1**, obținem **1 4 7.**

Acest exemplu numeric ne sugerează posibilitatea de a rezolva altfel problema anterioară:

- în locul numerelor considerăm variabilele **a, b, c**;

- comparând pe rând **a** cu **b**, **b** cu **c** și din nou **a** cu **b**, iar dacă este cazul interschimbăm conținutul acestor variabile.

```
întreg a, b, c, m;
Citește a, b, c
Dacă a > b atunci
   | m ← a
   | a ← b
   | b ← m
   |■
Dacă b > c atunci
   | m ← b
   | b ← c
   | c ← m
   |■
Dacă a > b atunci
   | m ← a
   | a ← b
   | b ← m
   |■
Scrie a, b, c
```

Problema 2. *Se citește* **n** *număr natural, strict mai mare ca* **1**. *Să se precizeze dacă este prim sau nu.*

Pentru rezolvare, o primă idee de lucru constă în a considera toți divizorii posibili ai numărului **n**. Dacă nici unul dintre aceștia nu-l divide, înseamnă că numărul este prim. Dar care sunt acești divizori?

- o posibilitate ar fi să considerăm divizorii cuprinși între **2** și **n-1**;

- o a doua posibilitate este de a considera divizorii cuprinși între **2** și jumătatea lui **n** (**n DIV 2**), soluție mult mai avantajoasă decât prima, întrucât avem mai puțini divizori și implicit se calculează mai puțin;

- o a treia posibilitate constă în a considera toți divizorii cuprinși între **2** și parte întreagă din radical din **n** (presupunând cunoscută teorema care afirmă că, dacă un număr nu are nici un astfel de divizor, el este prim).

Ultima variantă este mai bună. Spre exemplificare, vom considera numărul **999**; în prima variantă trebuie verificați **997** divizori, în a doua **447** iar în a treia numai **29** de divizori. Corespunzător acestei ultime variante redactăm algoritmul:

```
întreg n, i
logic prim;
Citește n
prim ← true
```

```
Pentru i←2,⌊√n⌋ execută
   | Dacă n mod i = 0 atunci prim ← false
   |    |■
   |■
Dacă prim atunci Scrie  "numărul este prim"
   |      altfel Scrie  "numărul nu este prim"
   |■
```

Algoritmul prezentat are un neajuns. Să presupunem că dorim să verificăm dacă numărul **1000** este prim. Primul divizor posibil (**2**) divide acest număr. Cu toate acestea, în loc să se afișeze rezultatul imediat, se verifică și ceilalți divizori posibili. Trebuie găsită o modalitate prin care, la găsirea unui divizor, să se afișeze faptul că numărul nu este prim.

```
întreg n, i;
Citește n
i ← 2
cât timp i < n  și  n mod i ≠ 0 execută
  | i ← i+1
  |■
Dacă i = n atunci Scrie "numărul este prim"
  |         altfel Scrie "numărul nu este prim"
  |■
```

Acest algoritm are dezavantajul că, atunci când numărul este prim, se încearcă toți divizorii posibili cuprinși între **2** și **n-1**. Puteți găsi un algoritm care să elimine acest dezavantaj?

Problema 3. *Să se scrie un algoritm care afișează primele* **nr** *numere prime,* **nr** *fiind citit de la tastatură.*

Se pune problema să asigurăm un ciclu în care să se caute numerele prime printre numerele **2**, **3**, ș.a.m.d., până se afișează cele **nr** cerute.

```
întreg n, i, j, nr, găsit
Citește nr
n ← 2
j ← 0
Repetă
   | găsit ← 1
   | Pentru i ← 2, ⌊⌊√n⌋⌋ execută
   |   | Dacă n mod i = 0 atunci găsit←0
   |   |   |■
   |   |■
   | Dacă găsit=1 atunci
   |   | Scrie n
   |   | j ← j+1
   |   |■
   |  n ← n+1
   |■
    până când j = nr
```

Problema 4. *Se citește* `n`, *număr natural* `>1`. *Să se descompună în factori primi.*

Algoritmul constă în următoarele:

- se citește `n`;
- se inițializează primul divizor posibil (`i=2`);
- secvența următoare se repetă până când `n=1`:
  - ◊ se inițializează `fm` cu `0` (factor de multiplicitate, arată puterea la care se găsește factorul prim);
  - ◊ atât timp cât `i` îl divide pe `n`, se execută următoarele
    - se mărește cu `1` factorul de multiplicitate;
    - se împarte `n` la `i`;
  - ◊ dacă `fm` este diferit de `0` (a fost găsit divizor al lui `n`) se afișează `i` și `fm`;
  - ◊ se adună `1` la `i` (se incrementează).

Să presupunem că am citit `n = 12`. Algoritmul decurge astfel:

- se inițializează `i` cu `2`;
- se inițializează `fm` cu `0`;
- întrucât `2` divide pe `12`, `fm` va lua valoarea `1`, iar `n` valoarea `6`;
- întrucât `2` divide pe `6`, `fm` va lua valoarea `2` și `n` valoarea `3`;
- `2` nu divide pe trei, se trece mai departe;
- `fm` este diferit de `0`, se afișează `2` 'la puterea' `2`;
- se mărește `i` cu `1`;
- ...
- `n` este `1` și algoritmul se încheie.

```
întreg n, i, fm;
Citește n
i ← 2;
Repetă
  | fm ← 0;
  |  Cât timp n mod i = 0 execută
  |     |    fm  ←  fm+1;
  |     |    n   ←  n div i
  |     |■
  |  Dacă fm ≠ 0 atunci
  |    |  Scrie i, " la puterea " fm
  |    |■
  | i ←  i+1
  |■
Până când n = 1
```

Problema 5. *Să se afișeze toate numerele naturale mai mici sau egale cu* **10000** *care se pot descompune în două moduri diferite ca sumă de cuburi.* Exemplu: $1729=1^3+12^3=9^3+10^3$.

Într-o structură repetitivă de tip **Pentru...execută** se generează, pe rând, numerele $n \in [1, 10000]$. Presupunând că **n** se descompune în i^3+j^3, este evident faptul că $i,j \leq \sqrt[3]{n}$. Pentru a nu găsi o pereche **(i,j)** de două ori (de exemplu **3,5** și **5,3**), $i \in [1, \sqrt[3]{n}]$, iar $j \in [i, \sqrt[3]{n}]$ Variabila **nr** reține numărul de perechi **(i,j)** găsite.

```
întreg n, nr, max, i, j, i1, j1, i2, j2;
Pentru n ← 1, 10000 execută
|   max ← [∛n]
|   nr ← 0
|   Pentru i ← 1, max execută
|   |   Pentru j ← i, max execută
|   |   |   Dacă i*i*i+j*j*j = n atunci
|   |   |   |   Dacă nr = 0 atunci
|   |   |   |   |   i1 ← i
|   |   |   |   |   j1 ← j
|   |   |   |   altfel
|   |   |   |   |   i2 ← i
|   |   |   |   |   j2 ← j
|   |   |   |   |■
|   |   |   |   nr ← nr+1;
|   |   |   |■
|   |   |■
|   |■
|   Dacă nr = 2 atunci
|   |   Scrie n, i1, j1, i2, j2
|   |■
|■
```

2.4 Scheme logice (facultativ)

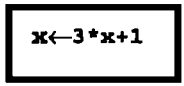
Există și o altă formă de redactare a algoritmilor, prin scheme logice. Pentru aceasta trebuie să cunoaștem următoarele:

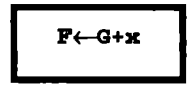
1. Orice schemă logică începe cu **Start** și se termină cu **Stop**.

✓ În programarea structurată se cere să utilizăm o singură dată un astfel de simbol.

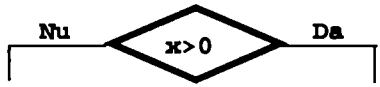
2. Pentru a simboliza o operație de citire sau una de scriere se folosește un paralelogram, așa cum sunt cele de mai jos:

3. Pentru a simboliza operații de calcul (atribuiri) se utilizează dreptunghiuri:

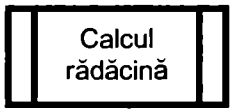


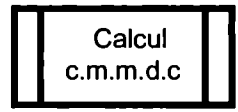


4. Pentru a verifica dacă o anumită condiție este îndeplinită se utilizează rombul. În interior se scrie expresia logică care cuantifică dacă condiția este sau nu îndeplinită (în funcție de valoarea produsă de expresia logică). Dacă expresia logică produce valoarea **True**, pe ramura respectivă vom scrie **Da**, iar dacă produce valoarea **False**, pe ramura respectivă scriem **Nu**.



5. Pentru a cuantifica o prelucrare complexă, care, eventual, va fi detaliată prin altă schemă logică (subschemă) utilizăm simbolul următor:

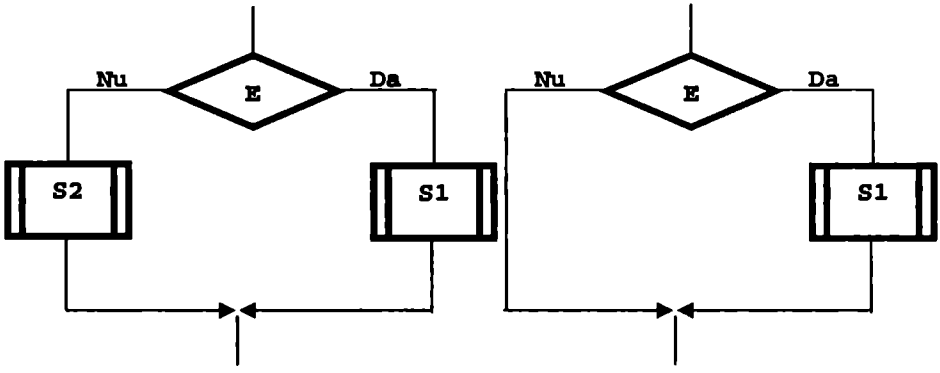




6. Toate blocurile sunt unite prin drepte. Pentru un set de date de intrare se parcurge un drum de la **Start** către **Stop** executându-se toate operațiile aflate în blocurile care se întâlnesc în drum.

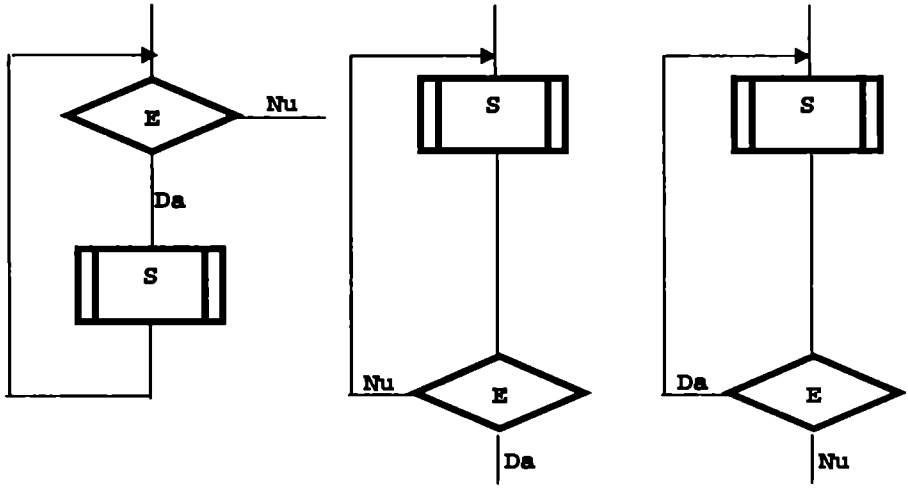
În aceste condiții prezentăm subschemele reprezentate de principalele structuri (Revedeți-le!)

A. Structura alternativă.



Forma completă Forma simplificată

B. Structuri repetitive.



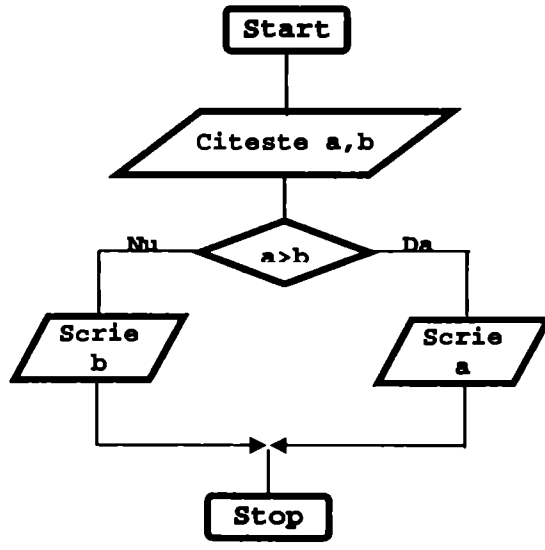
`Cât timp...execută` `Repetă...până când` `Repetă...cât timp`

✓ Structura **`Pentru...execută`** nu are corespondent direct în schemele logice, dar ea poate fi redată cu ajutorul celorlalte structuri.

În continuare, vom da un singur exemplu de problemă pentru care prezentăm algoritmul prin schemă logică.

Se citesc două numere reale. Să se afișeze numărul cel mai mare (valoarea maximă):

Prin utilizarea schemelor logice algoritmii se înțeleg mai ușor (se spune că un desen este mai util decât o mie de vorbe...). Dar, redactarea algoritmilor prin scheme logice prezintă și dezavantaje, cum sunt:



a) spațiul mare ocupat pe hârtie de o schemă logică (imaginați-vă cum arată schema logică a unui algoritm cu patru instrucțiuni repetitive imbricate).

b) redactarea cu dificultate a algoritmilor recursivi (îi veți studia în anul următor) sau a programării pe evenimente (utilizată astăzi oriunde în lume).

Datorită acestor motive, în privința schemelor logice, părerile specialiștilor sunt împărțite. În această carte schemele logice sunt prezentate doar la nivel informativ. Totuși, nimeni nu vă oprește să exersați redactarea algoritmilor cu ajutorul lor.

Probleme propuse

1. Scrieți expresia funcției evaluată de algoritmul următor:

```
Real f,x
Citește x;
Dacă x<0 atunci
   | f ← x/2
   | altfel
   |    Dacă x=0 atunci
   |       | f ← 10
   |       | altfel
   |       |   f←x*x
   |       |■
   |■
Scrie f;
```

2. Ce greșeală conține algoritmul următor?

```
Real f,x
Citește x;
Dacă x<0 atunci
   |   f ← -x
   | altfel
   |    Dacă x≥0 atunci
   |      | f ← x
   |      |■
   |■
```

3. Ce greșeală conține algoritmul următor?

```
Real f,x
Citește x;
Dacă x<-1 atunci
   |    f ← x*x
   | altfel
   |    f ← 1/x
   |■
```

4. Ce afișează algoritmul următor, dacă se citește **234**? Dar dacă se citește **78**? Puteți spune ce realizează acest algoritm?

```
întreg n,c
Citește n
Cât timp n≠0 execută
|   c ← c+1
|   n = ⌈n/10⌉
|■
Scrie c;
```

5. Care este efectul executării secvenței următoare, dacă se citește **n=7**? Dar dacă se citește **n=8**?

```
întreg n, a
Citește n
a=1;
Cât timp n≠a execută
|   a ← a+2
|■
Scrie a;
```

6. Stabiliți valoarea de adevăr pentru fiecare din propozițiile de mai jos (dacă este adevărată sau dacă este falsă).

a) Singura structură repetitivă indispensabilă este **Cât timp...execută**;

b) O structură repetitivă, urmată de altă structură alcătuiesc, în ansamblu, o structură liniară.

c) Ansamblul alcătuit dintr-o structură repetitivă inclusă într-o structură de tip **Dacă** este o structură repetitivă.

d) Un algoritm este, în ansamblul său, o structură liniară.

Scrieți algoritmii pentru rezolvarea următoarelor probleme:

7. Se citesc **a**, **b**, **c**, **d**, **e**, f numere naturale strict pozitive. Se cere să se scrie algoritmul care verifică relația de mai jos. Dacă ea este îndeplinită, se va va afișa **ADEVARAT**, altfel se va afișa **FALS**. Algoritmul nu va folosi împărțirea.

$$\frac{a}{b} \le \frac{e}{f} \le \frac{c}{d}$$

8. Se citește **x**, număr real. Să se evalueze expresia de mai jos:

$$F = \begin{cases} x^2 + x - 2, & pentru\ x \le 10, \\ \dfrac{1}{x}, & pentru\ 10 < x \le 20, \\ \dfrac{x-1}{x+1}, & pentru\ x > 20. \end{cases}$$

9. Se citesc **3** numere întregi. Să se afișeze (dacă există) un număr care este egal cu suma celorlalte două.

10. Se citesc **a** și **b** numere naturale. Câte numere pare se găsesc in intervalul închis **[a,b)**?

11. Se citește un număr natural **n**. Câte cifre are reprezentarea lui în baza **2**?

12. Se citesc **n,k∈N***, unde **k∈[2,9]**. Câte cifre are numărul **n** exprimat în baza **k**?

13. Se citește un număr natural. Să se afișeze produsul cifrelor lui. Exemplu: se citește **122** și se afișează **4**.

14. Se citește un număr natural format din **3** cifre. Să se afișeze numărul maxim care se obține cu cifrele sale. Exemplu: se citește **132** și se afișează **321**.

15. Afișați câtul și restul împărțirii întregi a două numere întregi citite. Atenție! Acestea pot fi și negative.

16. Ionel a depus la bancă o sumă de **s** lei. Dobânda este **p%** pe lună. Ce sumă are Ionel după **k** luni? Programul va citi **s, p, k** și va afișa suma cerută.

17. Se citește **n**, număr natural. Să se evalueze expresiile de mai jos:

$$E=\frac{1}{1}+\frac{1}{1\cdot 2}+\frac{1}{1\cdot 2\cdot 3}+\frac{1}{1\cdot 2\cdot 3\cdot 4}+\ldots+\frac{1}{1\cdot 2\cdot \ldots \cdot n}$$

$$E=1+1\cdot 3+1\cdot 3\cdot 5+\ldots+1\cdot 3\cdot \ldots \cdot (2\cdot n+1)$$

$E = E_1 + E_2 + \ldots + E_n$ unde E_i este dat de:

$$E_i=\frac{1}{1\cdot 2}+\frac{1\cdot 2}{1\cdot 2\cdot 3}+\frac{1\cdot 2\cdot 3}{1\cdot 2\cdot 3\cdot 4}+\ldots+\frac{1\cdot 2\cdot \ldots \cdot i}{1\cdot 2\cdot \ldots \cdot (i+1)}$$

18. Se citește un număr natural. Se poate descompune în sumă de pătrate? Exemplu: se citește **13**. Răspuns: **Da** ($3^2+2^2=13$).

19. Afișați toate numerele prime aflate între două numere naturale citite.

20. Se citesc, pe rând, **n** numere reale. Se cere produsul celor care sunt diferite de **0**.

21. Se citesc, pe rând, **n** numere reale. Se cere să se afișeze suma maximă care se poate forma cu ajutorul lor.

22. Se citesc, într-o ordine oarecare, numărul natural **n** și **n-1** numere distincte dintre numerele, **1,2.... n.** Algoritmul trebuie să decidă care număr natural nu a fost citit.

23. Se citesc, în ordine, cele **n** cifre ale unui număr natural. Se cere să se construiască și să se afișeze numărul natural format. Exemplu: se citesc **6, 7, 3**. Se va afișa **673**.

24. Se citește un număr întreg. Să se convertească într-o succesiune de litere după următorul algoritm:

lui **0** îi corespunde **a**;
lui **1** îi corespunde **b**;...

25. Se citește **x**, număr real **>2**. Se cer **p** și **q**, numere prime, astfel încât **p≤x<q** și diferența **q-p** este minimă.

Testele 26 și 27 se referă la algoritmul următor:

```
întreg n, s
Citește n
s←0;
Cât timp n≠0 execută
|   n ← n div 10
|   s ←s+1
|■
Scrie s;
```

26. Ce se afișează dacă se citește **021**?

a) **2** b) **3** c) **12** d) **21**

27. Pentru care dintre perechile de numere de mai jos algoritmul afișează același rezultat?

a) **21 9** b) **3 231** c) **121 13** d) **2 09**

Testele de la 28 la **30** se referă la algoritmul următor. O persoană alege un număr natural între **1** și **99** și calculatorul încearcă să găsească numărul. La fiecare pas, calculatorul afișează numărul pe care-l presupune că l-a ales persoana. Dacă numărul coincide cu cel ales de persoană, utilizatorul va tasta **0**. Dacă numărul afișat de calculator este mai mic decât numărul ales, persoana va tasta **1**, iar dacă acesta este mai mare decât numărul ales, persoana va tasta **2**. Se presupune că răspunsurile persoanei sunt corecte.

```
a←1; b←100;
Repetă
|  m←(a+b) div 2
|  Scrie m;
|  Citește Rasp
|    Dacă Rasp=1 atunci
|      |  a ← m
|      | altfel
|      |  b ← m
|      |■
|■
 Până când Rasp=0
```

28. Câte numere va afișa calculatorul, dacă numărul ales de persoană este **51**?

a) **8** b) **5** c) **6** d) **2**

29. Care este numărul ales de utilizator dacă acesta răspunde cu **1 2 1 0**?

a) 63 **b)68** **c) 52** **d) 76**

30. Care este numărul maxim de numere afișate de calculator la ghicirea unui număr?

a) 7 **b)8** **c) 6** **d) 4**

Testele de la 31 la 34 se referă la algoritmul următor, unde **div** este operatorul de împărțire întreagă și * este operatorul de înmulțire:

```
întreg n,n1,i
Citește n
Citește n1
i←10;
Cât timp (n div i)≠0 execută
|   i ←i*10
|■
n ← n*i+n1
Scrie n;
```

31. Ce se afișează dacă se citesc, în acestă ordine, **21** și **3**?

a) 2 **b)24** **c) 2103** **d) 213**

32. Pentru care dintre perechile de mai jos se nu se afișează **12**?

a) 1 2 **b)12 0** **c) 0 12** **d) 00 12**

33. Spunem că două perechi sunt echivalente dacă produc aceeași ieșire. Care dintre perechile de mai jos sunt echivalente?

a) (45 1) (450,1)
b) (12 12) (1200,12)
c) (1,5) (0,15)
d) (13,13) (1,313)

34. Care este modificarea care trebuie efectuată astfel încât algoritmul să efectueze concatenarea celor două numere citite? De exemplu, dacă se citește **44** și **1**, să se afișeze **441**.

Testele de la 35 la 37 se referă la algoritmul următor, unde **div** este operatorul de împărțire întreagă și **mod** este operatorul care returnează restul împărțirii a două numere întregi:

```
întreg n,s1,s2
Citește n
s1←0 s2←0
Cât timp n≠0 execută
      |    Dacă (n mod 2)=0 atunci
      |      |  s1 ← s1+1
      |      | altfel
      |      |  s2 ← s2+1
      |      |■
      |    n ← n div 10
      |■
Scrie s1, s2
```

35. Ce se afișează dacă se citește **12345**?

a) 2 3 **b)3 2** **c) 5** **d) 3 5**

36. Se citește un număr natural cu **5** cifre. Care dintre variantele de valori de mai jos nu poate fi niciodată rezultatul algoritmului, indiferent de numărul citit?

a) 5 0 **b) 0 5** **c) 2 3** **d) 3 3**

37. Considerăm că două numere sunt "de același tip" dacă pentru fiecare dintre ele, prin acest algoritm, se afișează aceleași valori în aceeași ordine! Care dintre perechile de numere de mai jos, conține numere "de același tip"?

a) 120 229 **b) 21 22** **c) 441 310** **d) 3 8**

Testele de la 38 la 40 se referă la algoritmul următor, unde **div** este operatorul de împărțire întreagă, **mod** este operatorul care returnează restul împărțirii a două numere întregi, iar * este operatorul de înmulțire.

```
întreg n,ninv,nsalv,s
Citește n
nsalv←n  ninv←0 s←0
Cât timp n≠0 execută
      | ninv←ninv*10+ n mod 10
      | n←n div 10
      |■
Cât timp (ninv≠0) si ((nsalv mod 10)=(ninv mod 10)) execută
      | s←s+1
      | ninv  ← ninv div 10
      | nsalv ← nsalv div 10
      |■
Scrie s
```

38. Ce se afișează dacă se citește **12431**?

a) 4 **b) 5** **c) 2** **d) 1**

39. Se citește **12321**. Ce se va afișa?

a) 1 **b) 5** **c) 3** **d) 4**

40. Care este valoarea maximă care se poate afișa prin citirea unui număr cu **n** cifre?

a) 1 **b) n-1** **c) n** **d)** $\left[\frac{n}{2}\right]$

Testele de la 41 și 42 se referă la algoritmul următor,

```
Pentru i ← 2, n execută
    | Pentru j ← 1, i-1 execută
    | | scrie j, i
    | |■
    |■
```

41. Dacă **n=10**, care dintre perechile de mai jos nu va fi afișată?

a) 4 3 **b) 2 10** **c) 1 10** **d) 8 9**

42. Câte perechi vor fi afișate dacă **n=9**?

a) 36 **b) 81** **c) 100** **d) 9**

Testele de la 43 și 44 se referă la algoritmul următor:

```
întreg a,b,i,n,s,s1
Citește a Citește b
s ← 0
Pentru i ← a, b execută
   | n ← i
   | s1 ← o
   | Cât timp n≠0 execută
   | | s1 ← s1+n mod 10
   | | n ← n div 10
   | |■
   | Dacă i mod s1 = 0  atunci
   |   |   s ← s + 1
   |   |■
   |■
Scrie s
```

43. Ce se afișează dacă **a=10** și **b=15**?

a) 0 **b) 1** **c) 2** **d) 3**

44. Ce se afișează dacă **a=1** și **b=9**?

a) 1 **b) 3** **c) 6** **d) 9**

Răspunsuri:

26. Algoritmul numără cifrele semnificative ale numărului citit **a) 27. d)**

28. c) 29. b) 30. a) La fiecare pas se înjumătățește intervalul în care se caută numărul. Trebuie găsit $\min\left\{n \middle| \frac{b-a}{2^n} \le 1\right\}$. **31. c) 32. b)**

33. c) 34. În loc de **Cât timp n div i execută** se va utiliza:
Cât timp n1 div i execută
35. a) Algoritmul numără cifrele pare și impare ale unui vector și le afișează în această ordine. **36. d) 37. a)**
38. d) Algoritmul afișează lungimea prefixului și sufixului care coincid pentru un număr natural citit. De exemplu, dacă numărul este **2232** lungimea este **1** (**2** este prima cifră, prefix, **2** este ultima cifră - sufix). Dacă numărul este palindrom lungimea prefixului coincide cu lungimea sufixului și este egală cu numărul de cifre ale numărului. Pentru rezolvare se construiește numărul oglindit și se compară cifră cu cifră ultimele cifre ale numărului și ale numărului oglindit. **39. b) 40. c) 41. a) 42. a)** În general se afișează $\frac{n(n-1)}{2}$ perechi. **43. c)** Algoritmul numără numerele din intervalul **[a,b]** care se divid cu suma cifrelor lor. **44. d)**

CAPITOLUL 3

Elemente de bază ale limbajului C++

3.1 Despre limbajul C++

La începutul anilor 70 a apărut limbajul **C** - creația lui Dennis Ritchie și Brian Kernighan. Limbajul **C++** -creația lui Bjarne Stroustrup- *poate fi privit* ca o *extensie a limbajului C care permite programarea pe obiecte*. Succesul avut pe piață l-a impus, deși există programatori (chiar dintre cei mai valoroși) care nu-l acceptă - se bazează pe ideea că limbajul **C** nu a fost creat pentru aceasta, prin urmare extensia este artificială. Noi vom învăța direct limbajul **C++**. Iată ce trebuie să știm, înainte de aceasta:

- Limbajul este extrem de flexibil - adică ne permite ca pornind de la un anumit algoritm, redactat în pseudocod, să putem scrie o mulțime de programe, care nu "seamănă" între ele, dar care fac același lucru. Cu alte cuvinte, o secvență poate fi codificată în multe feluri.
- Limbajul permite ca anumite secvențe să poată fi scrise "ermetic". Aceasta înseamnă că, după o anumită perioadă, chiar și cel care a scris secvența are dificultăți în înțelegerea ei. Este de la sine înțeles că nu trebuie să procedăm astfel.
- În același timp, programele se pot scrie și într-o manieră apropiată de limbajul pseudocod. Din acest motiv limbajul **C++** este preferat de mulți programatori. Pentru început, acesta este modul în care le scriem.
- În **C++** se poate usor greși, datorită faptului că este permis să scriem "aproape orice". Deci, atenție!

Pentru ca C++ să fie un limbaj portabil au apărut norme **ANSI C++** (standarde pentru limbaj). De fapt, de aceste standarde se ocupă o instituție specializată - **AMERICAN NATIONAL STANDARD INSTITUTE**. După cum am văzut, un limbaj conține, pe lângă setul de instrucțiuni, o mulțime de funcții care au rolul de a ajuta programatorul în elaborarea programelor. Atunci când o firmă (**Borland**, **Microsoft**) vinde un mediu de dezvoltare **C++**, toate funcțiile cerute de standarde trebuie să fie conținute de acesta. Pe lângă ele, firma poate introduce și alte funcții, dar utilizarea lor duce la crearea de programe neportabile.

3.2. Structura programelor C++

Un program C++ este alcătuit din una sau mai multe funcții. Ce înțelegem prin funcție?

Fie **A** algoritmul de rezolvare a problemei **P**. Prin respectarea structurilor clasice ale programării structurate (liniară, alternativă, repetitivă) algoritmul poate fi descompus în alți algoritmi mai mici, pe care convenim să-i numim subalgoritmi.

În linii mari, putem spune că:

- Fiecărui algoritm (subalgoritm) în C++ îi corespunde o funcție care codifică algoritmul (subalgoritmul) în instrucțiuni C++.
- O funcție poate întoarce un rezultat sau nu. Dacă aceasta nu întoarce nici un rezultat, spunem că tipul rezultatului este **void**. Tot așa, există funcții cu rezultat de tip întreg, real etc.
- O funcție poate conține declarații de variabile, care pot fi utilizate doar de ea, nu și de celelalte funcții.
- Programul C++ este alcătuit din una sau mai multe funcții, *din care una este rădăcină - adică nu poate lipsi și execuția programului începe automat cu ea.* Aceasta se numește **main**. De asemenea, programul poate conține și declarații de variabile globale - adică variabile cu care pot lucra toate funcțiile care îl alcătuiesc.

  Să vedem cum arată cel mai simplu program C++:

```
void main()
 {
 }
```

Programul de mai sus este corect. Cu toate acestea el nu face nici o operație. Dacă îl analizăm, putem trage primele concluzii.

- Este alcătuit numai din funcția rădăcină - **main**.
- Funcția rădăcină este de tipul **void** - adică nu întoarce nici un rezultat.
- Cele două paranteze rotunde **()** se atașează oricărei funcții. Între ele se scriu, opțional, anumiți parametri. În acest caz, aceștia lipsesc.
- Cele două acolade **{}** constituie corpul funcției. Între ele se scriu instrucțiunile care o alcătuiesc. În cazul de față mulțimea lor este vidă.

Observația 1. *Limbajul face distincție între literele mari și cele mici.* Programul de mai jos dă eroare de sintaxă, pentru că tipul **Void** nu este cunoscut.

```
Void main()
 {
 }
```

Observația 2. *Nu are importanță nici plasarea cuvintelor pe linie, nici spațiile dintre ele.*

Programul anterior poate fi scris și așa:

```
void main  (    )  {                }
```

Aceasta nu înseamnă că este bine să procedăm astfel. Să ne imaginăm că avem în față un program mare scris așa. El poate fi rulat, dar care programator va mai înțelege ceva, dacă îl privește?

Observația 3. *Este permis ca tipul funcției să lipsească.* În acest caz se presupune că funcția întoarce un rezultat întreg (de tipul **int**, după cum vom vedea). De exemplu, putem scrie și așa:

```
main()
{
}
```

În acest caz, la compilare, se primește un avertisment. De vreme ce tipul este **int**, se așteaptă ca funcția să întoarcă un rezultat de acest tip. Dar ea, nu întoarce nimic. Acest avertisment poate fi ignorat.

3.3. Descrierea sintaxei cu ajutorul diagramelor de sintaxă

Orice limbaj de programare se caracterizează prin sintaxă și semantică.

Sintaxa limbajului este dată de totalitatea regulilor de scriere corectă (în sensul acceptării sale de programul traducător (compilator) care are rolul de a converti programul în cod mașină pentru a fi executat). Dar un program corect din punct de vedere sintactic nu este automat un program bun. Corectitudinea sintactică este numai o cerință a programelor, tot așa cum pentru a fi campion mondial la alergări este necesar să nu ai picioarele amputate. Cel mai greu este ca programul să execute întocmai ce și-a propus cel ce l-a realizat.

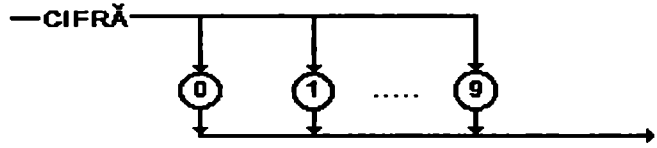
Prin semantica unui limbaj se înțelege semnificația construcțiilor sintactice corecte (de exemplu, ce anume realizează instrucțiunile, etc).

Sintaxa este formalizată perfect din punct de vedere matematic dar nu același lucru se întâmplă și cu semantica. Sintaxa poate fi descrisă cu ajutorul diagramelor de sintaxă sau a notației BNF. În continuare vom prezenta diagramele de sintaxă.

În ce constă o descriere cu ajutorul diagramelor de sintaxă? Să presupunem că dorim să descriem riguros cum arată un număr în baza 16. Dacă folosim o descriere de genul un număr în baza 16 are în componența sa cifre și litere, cifrele pot fi de la **0** la **9** iar literele **a, b, c, d, e, f, A, B, C, D, E ,F** și dăm câteva exemple, nu suntem suficient de riguroși.

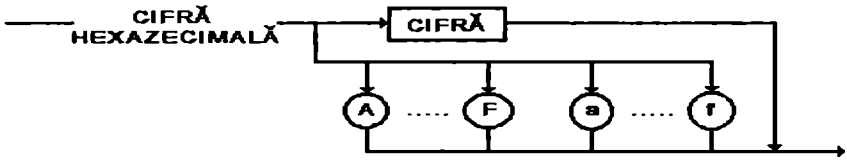
Imediat pot apărea întrebări de genul: **A** reprezintă un număr în baza **16**? Mai mult, dacă dorim să facem un program care să recunoască un număr citit este sau nu în baza **16**, situația se complică. Reluăm descrierea unui număr în baza **16**, cu ajutorul diagramelor de sintaxă.

O cifră este definită după cum se vede mai jos:

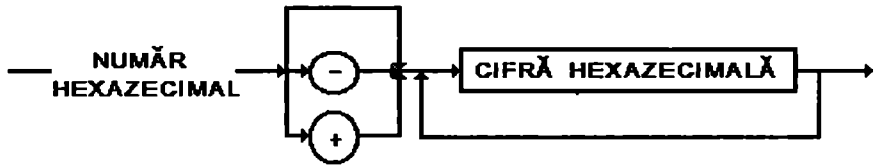


Se observă că o astfel de schemă are un nume, numele elementului sintactic definit (în cazul nostru cifră hexazecimală). Pentru a putea obține o cifră trebuie să urmez un drum prin acest desen (în matematică se numește graf orientat), de la intrare (din dreptul numelui) până la ieșire, urmând sensul săgeților. Problema se poate pune și invers: fiind dat un caracter oarecare să se precizeze dacă este sau nu cifră. Caracterul dat este cifră dacă în diagrama prezentată există un drum care trece printr-un cerculeț ce prezintă caracterul. Acum am definit riguros cifra.

Putem defini riguros cifra hexazecimală:



În noua diagramă apare un dreptunghi în care este scris cuvântul **CIFRĂ**. Dreptunghiul se utilizează atunci când se folosește o noțiune ce a fost definită printr-o altă diagramă de sintaxă. Orice drum folosim în acest desen (de la intrare la ieșire) obținem sau o cifră sau una din literele care pot face parte din structura unui număr hexazecimal:



În acest ultim desen observăm că este permisă de mai multe ori trecerea prin același loc.

În esență, o diagramă de sintaxă este constituită din următoarele simboluri grafice:

- cercuri - încadrează anumite simboluri speciale;
- elipse - încadrează cuvinte rezervate (nu pot fi folosite în alt context);

- dreptunghiuri - încadrează elemente definite prin alte diagrame de sintaxă.
- săgeți - indică sensul de parcurgere a diagramei.

Orice drum de la intrare la ieșirea din diagramă reprezintă o construcție sintactică corectă.

Ne-am putea întreba care este motivul unei descrieri atât de riguroase? Un program scris într-un limbaj de programare trebuie tradus de către programul numit compilator în cod mașină. În primul rând programu trebuie să fie corect din punct de vedere sintactic (altfel nu poate fi tradus) A verifica corectitudinea nu este un lucru simplu (pot fi foarte multe forme de programe). Din acest motiv, verificarea se face după reguli care trebuie descrise foarte clar, iar această descriere este realizată de diagramele de sintaxă.

3.4. Vocabularul limbajului

Vocabularul oricărui limbaj este format din:

- setul de caractere;
- identificatori;
- separatori;
- comentarii.

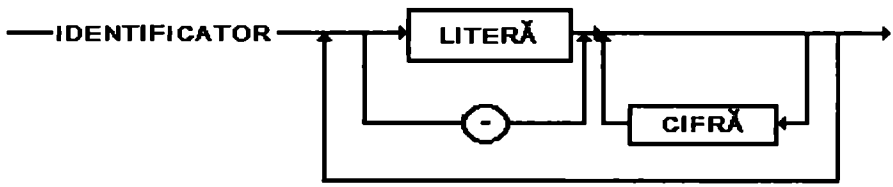
Setul de caractere. Reprezintă ansamblul de caractere cu ajutorul cărora se poate realiza un program C++. Acesta este alcătuit din:

- litere mari și mici ale alfabetului englez (**A-Z,a-z**);
- cifrele sistemului de humerație în baza **10** (0–9);
- caractere speciale: +, –, *, /, =, ^, <, >, (,), [,], {, }, ., ,, :, ;, #, $, @, _, și **blank** (spațiu).

Identificatori. *Prin identificatori înțelegem o succesiune de litere, cifre sau caracterul special '_' din care prima nu trebuie să fie cifră.* Cu ajutorul identificatorilor se asociază nume constantelor, variabilelor, procedurilor etc.

Exemple de identificatori: **a1**, **tasta**, **un_numar**, **_variabila**.

Contraexemple: **1ar**, **mt&** (primul începe cu o literă, al doilea conține un caracter special). Diagrama de sintaxă a unui identificator este:



O categorie specială de identificatori este dată de cuvintele cheie ale limbajului (au un înțeles bine definit și nu pot fi folosite în alt context). De exemplu: **for**, **while**, pentru că acestea sunt nume de instrucțiuni.

Separatori și comentarii. Cele mai simple elemente alcătuite din caractere cu semnificație lingvistică poartă denumirea de unități lexicale. Acestea se separă între ele, după caz, prin unul sau mai multe blancuri, caracterul **CR**, sfârșit de linie sau caracterul '**;**'. Pentru ca un program să fie ușor de înțeles se folosesc comentariile. Acestea se plasează oriunde în program. Un comentariu poate fi scris în două feluri:

1. Între perechile de caractere '**/***' și '***/**'. Un astfel de comentariu poate ocupa mai multe linii ale textului sursă. Exemplu: **/* acesta este un comentariu */**.

2. În cazul în care dorim să scriem un comentariu pe o singură linie sursă, tastăm două caractere '**//**'. Exemplu: o linie poate conține:

```
a=2;// aceasta este o atribuire
```

Comentariul începe după cele două caractere '**//**'.

3.5. Citiri, scrieri

Să analizăm programul de mai jos, care citește un număr întreg (pe care noi îl introducem de la tastatură) și îl tipărește:

```
#include <iostream.h>
void main()
{
 int a;
 cin>>a;
 cout<<a;
}
```

În cazul în care introducem numărul **10**, calculatorul va tipări **10**, dacă introducem **16**, calculatorul va tipări **16** ș.a.m.d.

După cum intuiți, **a** este o variabilă de tip întreg. În **C++** citirile / scrierile se fac cu ajutorul unor funcții speciale. Mai mult, acestea au și o modalitate cu totul specială de apel. Pentru moment, învățăm să le utilizăm, reținând, totuși, că acestea au fost realizate cu ajutorul programării orientate pe obiecte. Reținem și faptul că pentru a putea efectua citiri/scrieri trebuie utilizată directiva:

```
#include <iostream.h>
```

De ce? Ca să utilizeze funcțiile cerute, compilatorul are nevoie de anumite informații referitoare la ele. Acestea se găsesc într-un fișier **header** (antet) numit **iostream.h**. Prin directiva **#include** se solicită ca fișierul **iostream.h** să fie inclus înaintea textului care reprezintă programul **C++**. Fișierul obținut nu va fi vizualizat - adică programatorul nu vede nimic pe monitor. După ce a fost efectuată această operație se trece la compilarea propriu-zisă. Această operație (mai corect fază), efectuată înaintea compilării, se numește precompilare sau preprocesare.

Pentru a realiza citirile folosim **cin>>**. Iată forma generală:

cin>>a_1>>a_2>>...>>a_k;

unde, a_1, a_2,...,a_k sunt variabile de un tip oarecare. Efectul este:

- se citește variabila a_1;
- se citește variabila a_2,
- . . .
- se citește variabila a_k,

Observații:

1. În cazul citirii mai multor variabile, la introducerea datelor se procedează astfel: se introduce valoarea care urmează a fi atribuită primei variabile, se tastează **Enter**, se introduce valoarea care urmează a fi atribuită variabilei următoare, se tastează **Enter**, ...

2. În situația în care pentru o anumită variabilă se introduce o valoare care nu coincide cu tipul ei - de exemplu, pentru o variabilă întreagă se introduce un șir de caractere - citirea se blochează (adică următoarele variabile nu mai sunt citite), iar calculatorul execută instrucțiunile următoare, fără a semnaliza aceasta.

Exemplu:

```
#include <iostream.h>
void main()
{
 int a,b,c;
 cin>>a>>b>>c;
}
```

- Dacă se introduce **1 Enter**, **2 Enter**, **32 Enter**, după citire, variabila **a** reține **1**, variabila **b** reține **2**, variabila **c** reține **32**.
- Dacă introduc **m Enter**, citirea se blochează. Variabilele **a**, **b**, **c** își păstrează conținutul nemodificat.

Pentru a realiza scrieri pe ecran folosim **cout<<**. Iată forma generală:

cout<<a_1<<a_2<<...<<a_k;

unde a_1, a_2,...,a_k sunt variabile sau constante. Constantele de tip șir de caractere se scriu între apostrof. Exemplu: **'un sir'**.

- se scrie variabila (constanta) a_1;
- se citește variabila (constanta) a_2,
- . . .
- se citește variabila (constanta) a_k,

Observații:

1. În mod normal, datele se scriu una după alta, fără a se lăsa spațiu între ele. Acesta din urmă este un caracter special. Exemplu: dacă **a**

și **b** sunt două variabile de tip întreg care reţin respectiv valorile **1** și **2**, atunci prin scrierea **cout<<a<<b;** se tipăreşte **12**.

2. Dacă se doreşte ca, la scriere, datele să fie separate prin unul sau mai multe spaţii, acestea se tipăresc separat. Pentru exemplul anterior punem: **cout<<a<<" "<<b;**

3. Dacă dorim ca după ce am scris ceva, ce urmează să fie scris pe rândul următor se foloseşte o constantă specială numită **endl**. În exemplul anterior, dacă dorim ca 1 să fie scris pe un rând și 2 pe rândul următor scriem: **cout<<a<<endl<<b;**

Observaţii generale.

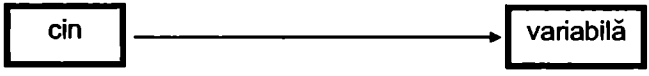
a. Atunci când programul citeşte o variabilă, este bine ca cel care introduce valoarea să ştie ce valoare trebuie să introducă. Tot aşa, atunci când programul tipăreşte o valoare, este bine ca valoarea să fie precedată de un text ce prezintă semnificaţia valorii tipărite. Analizaţi programul următor:

```
#include <iostream.h>
void main()
{
 int NrLat;
 cout<<"Nr laturi? ";cin>>NrLat;
 cout<<"Numarul laturilor este "<<NrLat;
}
```

b. Cum ţinem minte cele prezentate? "Numim" tastatura **cin**. Numim monitorul **cout**. La citire, datele *vin de la tastatură* în memorie. Deci punem **cin>>**. La scriere, datele trec din memorie *către monitor*, deci scriem **cout<<**. *Prin* **stream** *vom înţelege un flux de date de la o sursă către o destinaţie.*

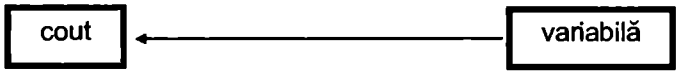
De ce este importantă noţiunea? *Noţiunea de stream nu face apel la tipul sursei şi la acela al destinaţiei.* Astfel, sursa poate fi: tastatura (**cin**), un fişier oarecare aflat pe *hard*, sau o variabilă a programului. Tot aşa, destinaţia poate fi: monitorul (**cout**), un fişier oarecare de pe *hard*, sau o variabilă a programului. Astfel, avem:

- stream-uri de intrare - *în care destinaţia este o variabilă a programului, iar sursa poate fi oarecare (excepţie o altă variabilă)* şi corespund clasicei noţiuni de *citire*. Noi am studiat numai cazul în care sursa este tastatura:



- stream-uri de ieşire - *în care sursa este o variabilă a programului (constantă), iar destinaţia oarecare (excepţie o altă*

variabilă) și corespund clasicei noțiuni de *scriere*. Noi am studiat numai cazul în care destinația este monitorul:



Un stream poate efectua orice tip de conversie permisă în C++. Despre ce este vorba? Intrările au un format specific, iar ieșirile altul - vezi **anexa 2**. De exemplu, dacă tipărim conținutul unei variabile de tip **int**, se face conversia din double către un șir de caractere (care apare pe ecran).

3.6. Tipuri de date, tipuri standard

Am văzut faptul că o variabilă poate reține date de un anumit tip. De exemplu, anumite variabile pot reține numere întregi, altele numere reale (la informatică asta înseamnă numere cu zecimale). Se pune întrebarea: *cum are programatorul posibilitatea să stabilească natura datelor care pot fi memorate de variabile?* Mecanismul este următorul: atunci când se declară o variabilă se precizează tipul ei.

Prin tip de date se înțelege:

- o mulțime de valori;
- o regulă de codificare a lor (modul în care se reprezintă în memorie);
- o mulțime de operații definite pe mulțimea valorilor.

Spunem că variabilele au un tip standard dacă acesta este cunoscut de către limbaj fără a fi definit în cadrul programului. Vom vedea că există posibilitatea ca programatorul să-și definească propriile tipuri, pentru ca apoi să declare variabile de tipul definit de el. Limbajul C++ conține următoarele tipuri standard:

1. tipuri întregi;
2. tipuri reale.

Observația 1. În C++ nu există tipul logic pe care l-am prezentat când am studiat limbaje de tip pseudocod. În acest caz, cum se pot efectua operațiile decizionale?

- Orice valoare diferită de **0**, a oricărui tip, este considerată ca fiind **TRUE**;
- Orice valoare **0**, a oricărui tip, este considerată ca fiind **FALSE**.

Observația 2. În C++ tipul caracter este asimilat tipurilor întregi. Ce înțelegem prin asta? Priviți programul de mai următor:

```
#include <iostream.h>
void main()
{
 char a;
 int b;
 a='y'; b=71;
 cout<<a<<"   "<<a+b;
}
```

Deși nu toate noțiunile care intervin au fost prezentate, o sumară analiză se poate face. Variabila **a** este de tip caracter, iar variabila **b** este de tip întreg. Variabilei **a**, i s-a atribuit caracterul **y**, iar variabilei **b** i s-a atribuit valoarea **71**. Se tipărește conținutul variabilei **a**, adică **'y'**, și ... **a+b**. Aparent această expresie este fără sens. Cum să adun un caracter cu un număr? În **C++** nu este nici o problemă. Se adună codul caracterului cu numărul și se tipărește suma. Deci tipul caracter este asimilat tipurilor întregi, pentru că această sumă se poate efectua.

3.6.1. Tipuri întregi

Tipurile întregi sunt:

Tipul **unsigned char**

- caracter fără semn, ocupă 8 biți și ia valori între **0** și **255**;

Tipul **char**

- caracter, ocupă 8 biți și ia valori între **-128** și **127**;

Tipul **unsigned int**

- întreg fără semn, ocupă **16** biți și ia valori între **0** și **65535**;

Tipul **short int**

- întreg scurt, ocupă **16** biți și ia valori între **-32768** și **32767**;

Tipul **int**

- întreg, ocupă *de regulă* **16** biți (numărul acestora diferă de la o implementare la alta) și ia valori între **-32768** și **32767**;

Tipul **unsigned long**

- întreg lung fără semn, ocupă **32** de biți și ia valori între **0** și **4.294.967.295**;

Tipul **long**

- întreg lung cu semn, ocupă **32** biți și ia valori între **-2.147.483.648** și **2.147.483.647**.

Variabilele de un tip întreg folosesc pentru memorarea datelor *codul complementar* (dacă se reţin date cu semn) sau baza 2 (atunci când se reţin date fără semn). Caracterele se memorează prin codul **ASCII**.

Declararea variabilelor de unul din tipurile întregi se face ca mai jos:

```
char a='y',b=76,c ;
int d,e=178;
```

Variabilele **a**, **b**, **c** sunt de tipul char, iar **d** şi **e** sunt de tipul **int**. Observaţi faptul că, la declarare, variabilele pot fi şi iniţializate.

La tipărirea unei variabile de tip caracter se tipăreşte caracterul şi nu codul său - deşi la atribuire pot ataşa fie codul caracterului, fie caracterul.

În secvenţa de mai jos se tipăreşte de două ori caracterul 'c'. Variabila **a**, a fost iniţializată cu **99** (codul **ASCII** al caracterului **'c'**);

```
char a=99,b='c';
cout<<a<<"  "<< b;
```

Variabilelor (operanzilor) de tip întreg li se pot aplica mai mulţi operatori. Aceştia vor fi trataţi într-un paragraf separat.

3.6.2. Tipuri reale

Variabilele de tip real reţin numere cu zecimale. Exemple de astfel de numere: 6,82, 9,3, -8,6. În locul virgulei, în *C++* se foloseşte punctul. Astfel, numerele de mai sus vor fi notate: 6.82, 9.3, -8.6. Tipurile reale sunt:

Tipul **float**

- ocupă **32** biţi şi ia valori între 3.4×10^{-38} şi 3.4×10^{38};

Tipul **double**

- ocupă 64 biţi **64 bits** 1.7×10^{-308} şi $1.7 \times 10^{+308}$;

Tipul **long double**

- ocupă **80** biţi şi ia valori între 3.4×10^{-4932} şi $1.1 \times 10^{+4932}$.

Pentru memorarea datelelor, se foloseşte reprezentarea în *virgulă mobilă*. Vezi anexa 2. Reţineţi faptul că variabilele reale se tipăresc (**cout<<**) cu 6 zecimale.

3.7. Constante

Există mai multe tipuri de constante.

1. **Constante întregi.** Acestea se clasifică astfel:

- **zecimale** (în baza 10). Exemple: **23, 1239, 56**.
- **octale** (în baza 8). O constantă în baza **8** se declară precedată de un **0** nesemnificativ. Exemplu: **0123**. Se reține numărul întreg $123_{(8)}$.
- **hexazecimale** (în baza 16). Acestea sunt precedate de **0X** sau **0x**. Exemplu: pentru **0X1A2** adică $1A2_{(16)}$ sau **0x1a2** adică $1A2_{(16)}$.

O constantă întreagă poate lua valori între **0** și **4.294.967.295**. Atenție: o constantă întreagă este pozitivă! În cazul în care se folosește semn (de exemplu **-123**) avem o *expresie constantă* ('-' este operator), care este evaluată.

2. **Constante caracter.** Acestea se trec între două caractere apostrof (**'**). Exemple: **'A', '1', 'a'**. Memorarea lor se face utilizând tipul *char*. Se memorează codul **ASCII** al caracterului respectiv. De exemplu, pentru **'1'** se memorează $49_{(10)}$.

Există și o altă modalitate de declarare a constantelor caracter, sub formă de *secvențe escape*. O secvență escape începe prin caracterul '\' (backslash). Să considerăm o constantă caracter **'a'**. Codul său este $97_{(10)}=141_{(8)}=61_{(16)}$. Printr-o secvență escape, constanta se introduce prin codul său într-una din bazele **8** sau **16**. De exemplu, constanta **'a'** poate fi scrisă (echivalent) astfel: **'\141'** sau **'\x61'**. În cazul când se folosește codul scris în baza **16**, acesta este precedat de caracterul **'x'**. Programul de mai jos tipărește de 4 ori caracterul **'a'**.

```
#include <iostream.h>
void main()
{
 char x1=97 ,x2='\141',x3='\x61',x4='a' ;
 cout<<x1<<x2<<x3<<x4;
}
```

Ne putem întreba: care este rostul acestor secvențe escape? Există caractere care nu se pot declara clasic (între două caractere apostrof), pentru că nu le putem tasta. Fie caracterul **newline** (codul $10_{(10)}$). Acesta poate fi declarat **'\12'** sau **'\xa'**. Pentru anumite caractere, în **C++** există notații speciale sub formă de secvență escape. De exemplu, constanta **newline** se poate scrie și **'\n'**. Alte exemple:

- **backslash: '\\','\134','\x5c';**
- **apostrof: '\'','\47','\x27';**
- **bel '\a','\7','\x7';**
- **cr '\r','\15','\xd'.**

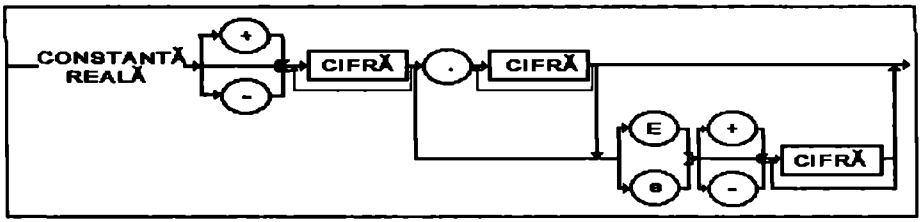
În mulțimea caracterelor care o alcătuiesc un rol aparte îl joacă *caracterele albe (whitespaces)*. Acestea sunt: blank (' '), tab orizontal (`\t`), tab vertical (`\v`), newline (`\n`), cr (`\r`). Acestea au un rol special pentru operațiile de citire / scriere. De exemplu, tab-ul orizontal poate fi folosit la scriere. Scrierea acestui caracter determină saltul cursorului cu 8 poziții.

```
#include <iostream.h>
main()
{
  char a='\t',b='i';
  cout<<a<<b;
}
```

Programul de mai sus tipărește: **bbbbbbbbi** (prin **b** am notat blank-ul).

Se admit și constante cu două caractere. De exemplu, '**AB**'. O astfel de constantă este memorată utilizând tipul **int**. Atenție: primul octet reține codul caracterului '**B**', iar al doilea codul caracterului '**A**'!

3. **Constante reale.** Exemplu **-45.66, 1., .2, 0.3, -2.5E-12,** adică -2.5×10^{-12}. Forma generală a unei constante de acest tip este:



4. **Constante șir de caractere.** După cum reiese și din denumire, cu ajutorul lor se reprezintă șiruri de caractere. O astfel de constantă se declară între două caractere ". Exemplu: "**acesta este un text**".

Pentru a da un nume constantelor folosim **const.** Forma generală a unei astfel de declarații este (construcția dintre paranteze drepte este opțională):

```
const [tip] nume=valoare;
```

Unde:

- **tip** - reprezintă tipul constantei (dacă este absent, tipul este **int**);
- **nume** - reprezintă numele constantei;
- **valoare** - reprezintă valoarea constantei.

Exemple:

- **const int numar=10;** Constanta de tip **int** numită **numar** are valoarea **10**.
- **const numar=10;** La fel ca mai sus;
- **const float pi=3.14;** Constanta de tip **float pi** are valoarea **3.14**.

Să analizăm mai atent modul de a da nume constantelor. Pentru aceasta considerăm ultimul exemplu. Dacă **const** ar fi lipsit am fi avut: **float pi=3.14;** adică am fi declarat o variabilă de tip **float** inițializată cu valoarea **3.14**. Prezența modificatorului **const** determină protecția la modificare a variabilei **pi**. Cu alte cuvinte, **pi** este o variabilă "citește numai" (are spațiu rezervat, reține o valoare care nu poate fi modificată).

Constantele întregi aflate în limitele **0** și **32767** pot căpăta nume și printr-un alt mecanism. Acesta are la bază un tip cunoscut de limbaj și numit **enum.** Să analizăm exemplele următoare:

- **enum timp {ieri, azi, maine};** Am declarat 3 constante **ieri, azi, maine,** inițializate implicit cu 0, 1, 2 (**ieri** are valoarea 0, **azi** are valoarea 1, **maine** are valoarea 2);

- **enum timp {ieri, azi=3, maine=7}; ieri** este inițializat cu **0**, **azi** este inițializat cu **3**, **maine** cu **7** (ultimele două inițializări sunt explicite);

- **enum timp {ieri, azi=3, maine=azi};** Este posibil ca două constante să fie inițializate cu aceeași valoare (**3** în exemplu);

- **enum timp {ieri=7, azi};** În cazul în care una dintre constante este inițializată, celelalte sunt inițializate implicit pornind de la valoarea acesteia la care se adaugă, pe rând, câte o unitate. În exemplu, constanta **azi** are valoarea **8.**

3.8. Expresii

3.8.1. Generalități

Definiție. *Se numește expresie o succesiune de operatori și operanzi legați între ei, după reguli specifice limbajului, în scopul efectuării unor operații (calcule, atribuiri, apelări de funcții etc.).* Operanzii pot fi: constante, variabile, funcții.

Esențial este să înțelegem modul în care se evaluează o expresie. Pentru aceasta prezentăm câteva noțiuni fundamentale.

- Prioritatea (precedența) operatorilor. Cu această noțiune suntem obișnuiți. Ea indică ordinea în care se efectuează operațiile. În C++ avem **16** niveluri de prioritate.

- Asociativitatea operatorilor. Noțiunea este nouă. Asociativitatea este de două feluri: *de la stânga la dreapta* și *de la dreapta la stânga.* De la început, precizăm faptul că operatorii cu aceeași prioritate au aceeași asociativitate.

Pentru a înțelege noțiunea de *asociativitate*, pornim de la o expresie în care operanzii sunt legați prin operatori cu aceeași prioritate. Dacă

asociativitatea operatorilor este de la stânga la dreapta, prima operație care se efectuează este cea corespunzătoare primului operator din stânga, a doua operație este cea corespunzătoare celui de-al doilea operator din stânga ș.a.m.d. Evident, în cazul în care *asociativitatea* este de la dreapta la stânga, prima operație care se efectuează este cea corespunzătoare operatorului din dreapta etc.

- Regula conversiilor implicite. *Fiind dat un operator binar (care leagă doi operanzi) în care unul este de un tip* (de exemplu **int**), *iar altul este de alt tip* (de exemplu **char**), *se cere să se precizeze tipul rezultatului.* Tipul acestuia este dat de *regula conversiilor implicite*, regulă prezentată în cadrul acestui capitol.

- O problemă nedecidabilă. *Fiind dat un operator binar, care va fi ordinea de evaluare a celor doi operanzi pe care îi leagă* (întâi operandul din stânga, apoi cel din dreapta sau invers)?

Există câțiva operatori (care vor fi prezentați la momentul potrivit) pentru care ordinea este garantată, dar nu și pentru majoritatea lor. Problema nu este legată exclusiv de cazul în care operatorii sunt funcții. Pe parcurs, vom da exemple în acest sens.

Reținem faptul că, *dacă scriem astfel de expresii, rezultatul depinde de compilatorul pe care îl avem la dispoziție, caz în care se obțin programe neportabile* (adică dacă sunt compilate cu un alt compilator, conduc la rezultate diferite).

- *Fiind dată o expresie se cere să se precizeze tipul ei.* În esență, trebuie stabilit tipul rezultatului obținut după evaluarea expresiei. Această problemă va fi prezentată pentru fiecare caz în parte. Pot avea rezultat de tip **int**, **float**, etc.

3.8.2. Operatori C++

Limbajul C++ este dotat cu un set puternic de operatori. Pentru ca în prezentare să nu facem tot timpul apel la prioritatea lor, ca și la modul în care aceștia se asociază (de la stânga la dreapta, sau invers), prezentăm de la bun început tabelul operatorilor C++.

prioritate	operator	asociativitate
1	()[]- >::.	*s* → *d*
2	!- + - + + - -*(*typecast*)*sizeof* *new* *delete*	*d* → *s*
3	.*- > *	*s* → *d*
4	*/%	*s* → *d*
5	+ -	*s* → *d*
6	<< >>	*s* → *d*
7	< <= > >=	*s* → *d*
8	== !=	*s* → *d*
9	&	*s* → *d*
10	^	*s* → *d*
11	|	*s* → *d*
12	& &	*s* → *d*
13	||	*s* → *d*
14	? :	*d* → *s*
15	=* =/ =+ =- =& =^ =|= <<= >>=	*d* → *s*
16	,	*s* → *d*

3.8.2.1 Operatori aritmetici

În C++ există următorii operatori aritmetici:

- - minus (unar, adică acționează asupra unui singur operand);
- + plus (unar);
- + (binar), pentru adunare;
- - (binar), pentru scădere;
- * (binar), are semnificația de înmulțire;
- / (binar), pentru împărțire;
- % (binar) restul împărțirii întregi.

Observații:

1. Operatorul `'/'` (împărțire) acționează în mod diferit în funcție de operanzi:
   a) dacă ambii sunt de tip întreg, rezultatul este întreg și are semnificația de împărțire întreagă. Cu toate acestea, rezultatul este corect (din punct de vedere matematic) numai dacă valorile care se împart sunt pozitive.
   b) dacă cel puțin un operand este de unul din tipurile reale rezultatul este real (se efectuează împărțirea obișnuită).
2. Operatorul `'%'` acționează numai asupra operanzilor de tip întreg. Rezultatul obținut este corect din punct de vedere matematic numai dacă ambii operanzi sunt numere naturale.
3. În cazul în care se împart două valori întregi se procedează astfel:
   a) se face împărțirea întreagă a celor două valori care sunt considerate în modul;
   b) semnul câtului se stabilește după regula semnelor (+ cu + rezultat +, + cu -, rezultat -) etc.
4. În cazul operatorului `'%'` se face împărțirea ca anterior (se obține C), iar restul se oține după formula R=D-ÎxC.
5. Se pot utiliza oricâte perechi de paranteze rotunde, pentru a impune ca anumite operații să se facă prioritar.

Exemple:

- Fie declarația: `int a=10;`. Atunci expresia `4*a/3` este de tip `int` și la evaluare se obține `13`. S-a înmulțit `4` cu conținutul variabilei `a` (`10`) și s-a obținut `40`. Apoi, `40/3=13` (a fost efectuată împărțirea întreagă). În aceleași condiții, expresia `4*(a/3)` are ca rezultat numărul `12`. La început s-a efectuat `a/3` și s-a obținut `3`, apoi `4*3=12`. Iată un exemplu în care se observă faptul că una este o expresie la matematică, și alta în `C++`.
- Fie declarația: `float a=10`. Expresia `4*(a/3)` are rezultatul `13.3333`, la fel ca expresia `4*a/3`. De această dată, s-a efectuat împărțirea a două numere reale.
- Fie declarația: `int a= -10;`. Atunci expresia `a%3` are ca rezultat numărul `-1`.
- Fie declarația: `int a=-10;`. Atunci expresia `a*-3` are ca rezultat `30`.
- Fie declarațiile: `int a=10; char b=2;float c=5;`. Atunci expresia: `a+b+c` are rezultatul `17.000000`.

În exemplu, apare o problemă foarte importantă: *care este tipul rezultatului, în cazul în care o expresie aritmetică are operanzi de mai multe tipuri?*

Dacă ținem cont de faptul că evaluarea unei expresii se face în pași (la fiecare pas se execută o operație aritmetică), întrebarea de mai sus se poate simplifica: *fie doi operanzi de tipuri oarecare* (întreg, real) *și un operator aritmetic* (de exemplu +), *care este tipul rezultatului expresiei a+b?*

Problema de mai sus se numește problema conversiilor aritmetice implicite (după cum vom vedea, conversiile se pot face și explicit).

Regula generală este: *se convertește unul din operanzi către tipul celuilalt (care poate reține rezultatul).* De exemplu, dacă unul din operanzi este de tip **int**, iar celălalt de tip **float**, rezultatul va fi de tip **float**.

P1 Iată pașii efectuați, în ordine, de către calculator:

1. Orice operand de tip **char** este convertit către tipul **int**.
2. Orice operand de tip **unsigned char** este convertit către tipul **int**.
3. Orice operand de tip **short** se convertește către **int**.

P2 Dacă un operand este de tip **long double**, atunci și celălalt operand se convertește către acest tip.

P3 Dacă un operand este de tip **double**, atunci și celălalt operand se convertește către acest tip.

P4 Dacă un operand este de tip **float**, atunci și celălalt operand se convertește către acest tip.

P5 Dacă un operand este de tip **unsigned long**, atunci și celălalt operand se convertește către acest tip.

P6 Dacă un operand este de tip **long**, atunci și celălalt operand se convertește către acest tip.

După execuția acestor pași, cei doi operanzi sunt de același tip, iar rezultatul va fi de tipul comun lor.

Atenție la conversii: în cazul în care avem doi operanzi de tipul **char** (**unsigned char**) rezultatul va fi de tip **int** (conform pasului 1). De exemplu, programul următor tipărește **97** și nu caracterul '**b**':

```
#include <iostream.h>
void main()
{
int a=0; char b='a';
cout<<a+b;
}
```

Revenind la ultimul exemplu, pentru calculul sumei se fac două conversii (la pasul 1 **char** în **int**, la pasul 4 **int** în **float**), iar rezultatul va fi de tip **float**.

3.8.2.2. Operatori relaționali

În C++ există următorii operatori relaționali:

- < (mai mic);
- <= (mai mic sau egal);
- > (mai mare);
- < (mai mare sau egal).

Întrucât în C++ nu există valorile logice **TRUE** și **FALSE**, rezultatul unei operații logice este **1**, în cazul în care inegalitatea este respectată și **0** în caz contrar.

Exemple.

- **3>5** - expresia ia valoarea **0**;
- **3<5** - expresia ia valoarea **1**;
- **3+7>=11-1** - expresia ia valoarea **1** (este posibil să scriem astfel de expresii, întrucât operatorii relaționali au o prioritate mai mică decât operatorii aditivi).

Operanzii pot fi constante, variabile, funcții care returnează tipuri numerice (pentru simplitatea expunerii au fost folosite constante întregi).

3.8.2.3. Operatori de egalitate

Aceștia sunt:

- == pentru egalitate;
- != pentru inegalitate.

Ca și operatorii relaționali, expresiile de acest tip returnează **0** sau **1** - în funcție de modul în care este respectată sau nu egalitatea (inegalitatea).

Exemple:

- **3==3**, rezultat **1**;
- **3!=3**, rezultat **0**;
- **3!=4**, rezultat **1**.

Operanzii pot fi constante, variabile, funcții.

3.8.2.4. Operatori de incrementare și decrementare

Acești operatori sunt unari și au rolul de a incrementa (adună `1`) sau decrementa (scad `1`) conținutul unei variabile. Operatorii sunt:

- `++` pentru incrementare;
- `--` pentru de decrementare.

Operatorii pot fi prefixați (aplicați în fața operandului) sau postfixați (aplicați după operand). Exemplu: fie `a` o variabilă de tip `int`.

Operatorul de incrementare prefixat aplicat variabilei `a` este `++a`;
Operatorul de incrementare postfixat aplicat variabilei `a` este `a++`;
Operatorul de decrementare prefixat aplicat variabilei `a` este `--a`;
Operatorul de decrementare postfixat aplicat variabilei `a` este `a--`.

Care este diferența între efectul unui operator aplicat prefixat sau postfixat? Să ne gândim la faptul că pot fi expresii care conțin și alți operatori în afara celor de incrementare (decrementare).

⇒ *Dacă operatorul este prefixat, variabila este incrementată (decrementată) înainte ca valoarea reținută de ea să intre în calcul.*

⇒ *Dacă operatorul este postfixat, variabila este incrementată (decrementată) după ce valoarea reținută de ea intră în calcul.*

Exemple:

- Fie `a` o variabilă de tip `int` care reține valoarea `1`. În urma evaluării expresiei `1+a++` se obține valoarea `2`, dar, după evaluare, `a` va reține valoarea `2` (variabila a fost incrementată după ce valoarea reținută de ea a intrat în calcul).
- La fel ca în cazul anterior, expresia `1-++a` produce valoarea `-1` (variabila a fost întâi incrementată, s-a obținut `2`, apoi s-a efectuat `1-2`).
- Expresia `1+++a` este eronată sintactic. Compilatorul a considerat că operatorul de incrementare este aplicat postfixat pentru `1` și nu prefixat pentru `a`. *Aplicarea unui operator de acest tip se poate face pentru o variabilă.*
- Dacă `a` și `b` sunt variabile de tip `int` care reține valorile `1` și `3` atunci expresia `a+++*b++` produce valoarea `3`, dar după evaluare cele două variabile reține `2` și `4`.
- În aceleași condiții, expresia `++a*++b` produce valoarea `8`, iar după evaluare `a` și `b` reține `2` și `4` (ca în cazul anterior).

Acțiunea operatorului de decrementare este similară celui de incrementare, cu diferența că se scade `1` din conținutul variabilei.

Operatorii de incrementare/decrementare se pot aplica și variabilelor de tip real.

În cazul utilizării formei postfixate, apar cazuri în care nu se poate decide care va fi valoarea produsă de expresie la evaluare.

Am precizat faptul că incrementarea se face după ce valoarea a intrat în calculul expresiei. Dar când? O posibilitate ar fi ca incrementarea să se facă imediat ce valoarea a intrat în calcul. Altă posibilitate este ca incrementarea să se facă după ce a fost calculată întreaga expresie. Sau s-ar putea să se facă între aceste două momente (după ce au fost efectuate deja câteva calcule). Specificațiile C++ nu prevăd cum se procedează în astfel de cazuri. Momentul când se face incrementarea poate prezenta importanță.

Fie expresia: `a+++a++` , unde a are valoarea inițială `1`. Transcriem etapele posibile ale evaluării:

1. La evaluare, se calculează `a+a=2`, iar după evaluare avem: `a=1+1=2`, `a=2+1=3` - expresia a dat rezultatul `2`.

2. Se procedează astfel: valoarea intrată în calcul pentru primul termen al adunării este `1`. Apoi, se incrementează `a` și se obține 2. Se face suma `1+2=3` (rezultat produs la evaluare) și `a` este din nou incrementată - se obține `3`.

Iată că se obțin rezultate diferite. Din acest motiv, vom evita scrierea unor astfel de expresii. Chiar dacă (întâmplător) programul funcționează așa cum dorim, el este neportabil (compilat cu alt compilator nu mai furnizează rezultatul dorit).

3.8.2.5. Operatori logici

Există trei operatori logici:

- ! negare logică;
- && și logic;
- || sau logic.

Acești operatori se aplică oricărei variabile și constante din cele învățate.

Operatorul *negare logică* acționează astfel: dacă operandul este o valoare diferită de `0`, rezultatul este `0`, altfel rezultatul este `1`.

Operatorul *și logic* (binar) acționează astfel: dacă ambii operanzi sunt diferiți de 0, rezultatul este `1`, altfel el este `0`.

Operatorul *sau logic* (binar) acționează astfel: dacă cel puțin unul din operanzi este o valoare diferită de `0`, rezultatul este `1`, altfel rezultatul e `0`.

Exemple:

- Dacă **a** este o variabilă de tip **float** care reține **2**, atunci **!a=0**, iar dacă **a** reține **0**, **!a=1**.
- Dacă **a** și **b** sunt două variabile de tip **int** care reține **1** și **3** atunci, **a&&b=1**, iar dacă **a** are același conținut, iar **b** reține **0**, **a&&b=0**;
- Dacă **a** și **b** sunt două variabile de tip **int** care reține **0** și **3** atunci **a||b=1**, iar dacă **a** are același conținut iar **b** reține **0** **a||b=0**.

Operatorii logici binari garantează modul în care se tratează operanzii: întâi cel din stânga, apoi (dacă este cazul) cei din dreapta. Astfel, dacă operandul din stânga este o valoare diferită de **0**, operatorul *sau logic* nu mai acționează asupra operandului din dreapta (este clar că rezultatul este **1**). Tot așa, dacă operandul din stânga este **0**, operatorul *și logic* nu mai evaluează operandul din dreapta (oricum rezultatul este **0**).

Datorită celor expuse anterior, putem avea surprize dacă lucrăm fără să fim atenți. Exemplu: **a&&b--**. Dacă **a** este **0**, nu se mai evaluează **b**, deci nu se va face decrementarea.

3.8.2.6. Operatori logici pe biți

Limbajul **C++** este dotat cu un set de operatori care permit accesul la bit (**vezi anexa 2**). Aceștia sunt:

- <<, >> operatori de deplasare;
- & și pe biți;
- | sau pe biți;
- ^ sau exclusiv pe biți;
- ~ negare pe biți (operator unar).

Acești operatori acționează numai asupra operanzilor de tip întreg.

Operatorul << este binar. El are rolul de a deplasa către stânga conținutul tuturor biților operandului din stânga sa, cu un număr de poziții egal cu valoarea reținută de al doilea operand. Pozițiile rămase libere (în dreapta) vor reține valoarea **0**. Dacă al doilea operand reține valoarea **m**, o astfel de deplasare este echivalentă cu înmulțirea cu 2^m (evident, dacă **m** este mai mic decât numărul de biți rezervat primului operand).

Operatorul >> este binar. El are rolul de a deplasa către dreapta conținutul tuturor biților operandului din stânga cu un număr de poziții egal cu valoarea reținută de al doilea operand. Dacă operandul din stânga este de un tip întreg fără semn, pozițiile rămase libere (în stânga) vor reține valoarea **0**. Dacă al doilea operand reține valoarea **m**, o astfel de deplasare este echivalentă cu împărțirea întreagă cu 2^m. În cazul în care primul operand este un întreg cu semn, fiecare poziție din stânga rămasă liberă se completează cu valoarea reținută de bitul de semn.

În cazul operatorilor binari &, |, ^, rezultatul se obține aplicând pentru fiecare pereche de biți aflați pe aceeași poziție regulile din tabelul următor. Atunci când cei doi operanzi nu au aceeași lungime (dar numai atunci - de exemplu, dacă ambii operanzi sunt de tip **char** și rezultatul este de tip **char**), se aplică regulile de conversie pentru expresii aritmetice.

OP_1	OP_2	$OP_1 \& OP_2$	$OP_1 \wedge OP_2$	$OP_1 \mid OP_2$
0	0	0	0	0
1	0	0	1	1
0	1	0	1	1
1	1	1	0	1

Operatorul ~ (negare pe biți) are rolul de a inversa conținutul biților (dacă un bit conține **0** va conține **1** și invers).

Exemple: Dacă **a** este de tip **int** și reține constanta hexa **000f**, iar **b** este de același tip și reține constanta hexa **0f03** atunci:

- **a&b = 0003 (în hexa);**
- **a|b= 0f0f;**
- **a^b= 0f0c;**
- **a<<2=003c;**
- **a>>2=0003;**
- **~a=fff0.**

3.8.2.7. Operatori de atribuire

Spre deosebire de limbajele de tip pseudocod, în **C++** atribuirea este operator. În plus, în **C++** avem mai mulți operatori de atribuire. Operatorul '=' se folosește într-o expresie de forma:

v=expresie

Aici, v este o variabilă.

Principiul de executare este următorul:

- *se evaluează expresia;*
- *varibilei v i se atribuie valoarea obținută (dacă este cazul se efectuează conversia respectivă).*

Se pot efectua și atribuiri multiple de forma:

v=v_1=v_2=.....=v_n=expresie

unde **v, v_1.....v_n** sunt variabile.

În acest caz principiul de executare este următorul:

- se evaluează expresia;
- valoarea obținută este atribuită variabilei **v_n** (eventual convertită - dacă este cazul);
- conținutul variabilei **v_n** este atribuit variabilei **v_{n-1}** (eventual se efectuează conversia necesară);
- . . .
- conținutul variabilei **v_1** este atribuit variabilei **v** (eventual se efectuează conversia necesară).

Am precizat faptul că acest operator se asociază de la dreapta la stânga. Ce am prezentat anterior evidențiază acest lucru.

Pentru atribuiri se mai pot utiliza și operatorii: *=, /=, %=, +=, -=, <<=, >>=, &>, ^=, |=.

O atribuire de forma: v op expresie, are același rezultat ca

v=v op expresie

(diferența este că, în primul caz, se generează un cod mașină eficient).

Observație. Am văzut faptul că atribuirea este operator care figurează într-o expresie. Ca orice expresie, aceasta are o valoare rezultată în urma evaluării. *În cazul atribuirilor valoarea rezultată este valoarea atribuită variabilei care este prima din stânga, după conversie (într-un șir de atribuiri).*

Exemple:

- Fie declarațiile: **int a=5, b=3; float c;**. În urma atribuirii **c=a/b**, **c** va conține valoarea reală **1.0**. Este normal să fie așa întrucât expresia **a/b** este de tip **int** (se efectuează împărtirea întreagă și se obține **1**), apoi această valoare este atribuită variabilei **c** (după ce este convertită).
- Fie declarațiile: **int a,b; float c;** și expresia **c=a=b=1**. După evaluare, **a** reține **1**, **b** reține **1**, **c** reține **1.0**, iar valoarea rezultată în urma evaluării este **1.0** (reală).
- Fie declarația: **int a=3;** și expresia **a*=2**. În urma evaluării, **a** reține valoarea **6** (**a←a*3** în pseudocod), iar expresia va produce valoarea de tip **int 6**.

Fie declarația `int a=0,s;`. Fie expresia: `s=(a=3)+(++a)`. Nu se poate decide dacă în urma evaluării `s` va reține 4 sau 7. De ce? Nimeni nu garantează care operand (din cei doi legați de operatorul +) va fi evaluat primul. Dacă este evaluat primul, atunci `a` va reține 3, iar expresia `a=3` va produce valoarea 3. Urmează evaluarea operandului următor: `++a`. În urma evaluării, `a` reține 4, iar expresia `++a` produce valoarea 4. În urma adunării, se obține 7. Altfel, să presupunem că primul operand evaluat este `++a`. Cum `a` este 0, acesta va reține 1, iar expresia produce valoarea 1. Apoi este evaluată expresia `a=3`. În urma evaluării, a reține 3, iar expresia produce valoarea 3. Prin urmare, se adună 1 cu 3, rezultatul fiind 4. Este un caz tipic când expresia nu poate fi evaluată precis.

Observație. În cazul în care unei variabile de unul din tipurile întregi i se atribuie conținutul unei variabile reale, acesta este trunchiat. Exemplu. Fie `a` o variabilă de tipul `int` și `b` o variabilă reală care reține `-1.9`. În urma atribuirii, `a` va reține `-1`. Atenție! Trunchiere nu înseamnă parte întreagă. Dacă variabilei `a` i s-ar fi atribuit partea întreagă a conținutului lui `b`, a ar fi trebuit să rețină `-2`.

3.8.2.8. Operatorul ',' (virgulă)

C++ permite programatorilor să scrie mai multe expresii separate prin virgulă ca mai jos:

$$exp_1, exp_2, \ldots, exp_n;$$

Întrucât, după cum rezultă din tabel, operatorul virgulă se asociază de la stânga la dreapta expresiile se evaluează în ordinea exp_1, exp_2, ..., exp_n. S-a convenit ca întreaga expresie (care cuprinde cele `n` expresii separate prin virgulă) să producă ca rezultat valoarea obținută în urma evaluării ultimei expresii (evident, tipul acestei valori este și tipul expresiei).

Exemplu. Fie declarațiile: `int a=1, b=5; float c;`

Expresia: `c=a=b+1,a=c+2,b=b+1` se evaluează astfel:

- se efectuează atribuirea multiplă. Astfel: `b+1=6; a=6;c=6;`
- `a=6+2=8;`
- `b=5+1=6`.

Expresia (în ansamblu) este de tipul `int` și produce valoarea 6.

3.8.2.9. Operatorul condițional

Se folosește în expresii de genul:

$$exp_1 ? exp_2 : exp_3$$

Principiul de executare este următorul:

- se evaluează exp_1;
- dacă aceasta produce o valoare diferită de 0, se evaluează exp_2 și exp_3 este ignorată (nu se evaluează);
- altfel, se evaluează exp_3 și exp_2 este ignorată.

În ansamblu, expresia este de tipul lui exp_2 sau exp_3 și produce valoarea exp_2 sau exp_3 (în funcție de cea care se evaluează).

Exemplu:

- Programul următor citește x (de tip `float`) și tipărește |x|. Cum a fost realizat? Se testează dacă x este mai mare sau egal cu 0. În cazul în care condiția este îndeplinită, se evaluează x (a doua expresie). În acest caz, expresia condițională, în ansamblul ei, ia valoarea pe care o reține variabila x. Această valoare se tipărește. Dacă x este strict mai mic decât 0, expresia evaluată este -x și aceasta este valoarea pe care o ia expresia condițională și care este tipărită.

```
#include <iostream.h>
main()
{ float x;
 cout<<"x=";;cin>>x;
 cout<<(x>=0?x:-x);
}
```

3.8.2.10. Operatorul sizeof

Are rolul de a returna numărul de octeți utilizați pentru memorarea unei valori. Operatorul `sizeof` poate fi utilizat într-una din cele două forme prezentate în continuare:

sizeof (expresie)

sizeof (tip)

Este de reținut faptul că *expresia nu este evaluată* (deci nu apar efecte secundare).

Exemple:

- Fie declarația: `int a; sizeof (a)=2` (dacă în versiunea C++, pe care o utilizăm, tipul `int` ocupă 2 octeți);
- Fie declarația: `float a; sizeof (a)=4;`

- `sizeof (float)=4;`
- `sizeof (a+b)=8,` unde `a` este de tip `int` și `b` este de tip `double`;
- `sizeof (++x)=4,` dacă x este de tip `float,` dar `x` rămâne nemodificat (expresia `++x` nu a fost evaluată).

3.8.2.11. Operatorul de conversie explicită.

De multe ori, dorim ca unul sau mai mulți operanzi să intre în calcul convertiți așa cum dorim (nu implicit). Pentru aceasta, înaintea operandului se trece între paranteze tipul său.

Exemple:

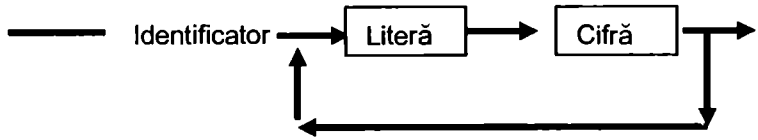
- Fie declarația: `float x= -1.9;` Atunci: `(int)x=-1` (se face conversia din `float` în `int` prin trunchiere);
- În aceleași condiții `(int)(++x+3)=2 (x=-1.9, ++x=-0.9, ++x+3=2.9, int (2.9)=2;)`
- Fie declarațiile `int a=3, b=6;`. Atunci `(float) a/b=0.5.`

Probleme propuse

1. Care dintre programele C++ următoare sunt corecte?

```
a)                    b)
main                  Main()
{                     {
}                     }
c)                    d)
main()                main() {  }
{
}
```

2. Dacă într-un limbaj ipotetic, un idendificator se definește prin următoarea diagramă de sintaxă, care dintre cuvintele de mai jos sunt identificatori? Semnificația blocurilor `Literă` și `Cifră` sunt cele din C++.



`a) m12a; b) mama; c) t1t2; d) 1a2b`

3. La executarea programului următor se introduc, pe rând, **2** și **3** (după fiecare introducere se tastează **Enter**). Ce va fi afișat după executare?

```
#include <iostream.h>
main()
{
 int a;
 cin>>a>>a;
 cout<<a<<endl<<a;
}
```

a) 2 3
b) 2
   3
c) 3
   3
d) 3 3

4. Ce se va afișa în urma executării secvențelor (programelor) următoare?

a)
```
#include <iostream.h>
main()
{unsigned char x=3;
 x=-x;
 cout<<(int)x;
}
```

a) 255
b) 0
c) 252
d) 3

b)
```
#include <iostream.h>
main()
{ unsigned char x=5;
 x=x<<3;
 cout<<(int)x;
}
```

a) 0
b) 2
c) 40
d) 5

c)
```
#include <iostream.h>
main()
{unsigned char x=5;
 x=x>>1;
 cout<<(int)x;
}
```

a) 0
b) 2
c) 5
d) 1

d) Se introduce de la tastatură **14** și **Enter**.

```
...
char x;
cin>>x;
cout<<x;
```

a) 1
b) 4
c) 14
d)

e)
```
cout<<(7>10);
```

a) eroare de execuție
b) 7>10
c) 0
e) 1

f)
```
#include <iostream.h>
main()
{ float x=1/2;
 cout<<x; }
```

a) 0
b) 0.5
c) Eroare
d) 1/2

g)
```
cout<<-15/4<<' '<<-15%4 ;
```

a) -3 -3
b) -4 2
c) -4 -2
d) eroare

5. Fie `x` și `y` două variabile logice. Avem formulele (de Morgan):

a) `!(x || y)=!x && !y.`
b) `!(x && y)=!x || !y.`

Demonstrați cele două formule, extrem de utile atunci când se neagă anumite expresii logice.

6. Cum scriem în C++ expresiile următoare?

$a)\ E=\frac{1}{2}+\frac{1}{3}+\frac{1}{4};$

$b)\ E=\frac{1}{2\cdot 3}+\frac{1}{3\cdot 4}+\frac{1}{4\cdot 5};$

$c)\ E=\frac{1+2}{3+4}+\frac{5+6}{7+8}+\frac{9+10}{11+12};$

$$d)E=3\cdot\frac{4+\dfrac{6+\dfrac{7}{8}}{\dfrac{5}{3}}}{\dfrac{3+\dfrac{1}{3}}{4+1{,}5\cdot\dfrac{8}{7}}}.$$

✓ Expresiile vor fi transpuse identic, fără simplificare. Acesta este numai un exercițiu de transcriere. Atunci când veți scrie programe, este bine să simplificați voi expresiile. În acest fel, calculatorul nu va efectua la fiecare rulare a programului calcule inutile.

7. Evaluați expresiile următoare precizând rezultatul și tipul expresiilor:

a) `x+1.5`, unde `x` este o variabilă întreagă ce conține numărul `7`;
b) `x<=4`, unde `x` este o variabilă întreagă ce conține numărul `4`;
c) `a+1>3`, unde `a` este o variabilă reală care conține numărul `3`;
d) `6/2;`

e) `x/y+y/x` unde `x` și `y` sunt variabile întregi care conțin, respectiv, `1` și `6`;
f) `x<3 || x>6` (unde `x` este o variabilă reală care reține `7`).
g) `x<3 && x>6` (la fel ca mai sus).

8. Fie `x` o variabilă de tip `int`. Pentru ce valori ale lui `x` expresia: `x+7==8*x` ia valoarea `1`?

9. La fel ca mai sus, numai că expresia este: `2*x+3==6`.

10. Ce erori conțin expresiile de mai jos?

a) `(x+3`, unde `x` este o variabilă întreagă.
b) `(x+1/((x+2)-1` (`x` este variabilă întreagă).
c) `y=5x` (`x, y` sunt variabile reale);

Pentru fiecare din testele care urmează, un singur răspuns este adevărat. Care este el? Justificați-l. Puteți folosi tabelul de priorități al operatorilor, ca, de altfel, orice documentație. Asta nu trebuie să mire pe nimeni. Un programator nu este o persoană care memorează, ci una care gândește. De asemenea, în cazul unor nelămuriri, testați răspunsul dvs. printr-un mic program.

11. Considerăm `long a=3; char b=2;` Care este valoarea produsă de expresia `a+b`, și care este tipul rezultatului?

a) Nu se poate aduna conținutul unei variabile de tip `char` cu acela al unei variabile numerice. Compilatorul furnizează eroare de sintaxă la compilare.
b) Rezultatul este `5` și este de tip `char`.
c) Rezultatul este `5` și este de tip `long int`.

12. Considerăm `int a=-10,b=3;` Care sunt valorile produse de expresiile `a/b` și `a%b` ?

a) `-3,33333...3` și `2;`
b) `-3` și `-1;`
c) `-4` și `2.`

13. Considerăm `int a=2;` Care este valoarea rezultată în urma evaluării expresiei `1/(2*a)`?

a) `0.25;`
b) Operatorul "`/`" acționează numai asupra variabilelor (constantelor) de un tip real. Avem eroare de sintaxă.
c) `0.`

14. Considerăm `int a=2,b=5;` Care este valoarea rezultată în urma evaluării expresiei `a+3<=b+2`?

a) `1`;
b) `0`;
c) Este obligatoriu să folosim paranteze, pentru că altfel nu se poate face evaluarea.

15. Considerăm `int a=2,b=5;` Care este valoarea rezultată în urma evaluării expresiei `1==a<b`?

a) `1` nu este egal cu `a`, rezultat `0`, `0` este < `b` (`b=5`) rezultat `1`;
b) `a` mai mic ca `b` rezultat `1`, `1` este egal `1`, rezultat `1`;
c) a<b rezultat `TRUE`, iar `1` nu este egal cu `TRUE`, rezultat `0`.

16. Considerăm `int a=5,b=5;` Care este valoarea rezultată în urma evaluării expresiei `a++/++b`?

a) `1;`
b) `0,833333;`
c) `0.`

17. Considerăm `int a=5,b=5;` Care este valoarea rezultată în urma evaluării expresiei `!a+b`?

a) `5;`
b) `10;`
c) `0.`

18. Considerăm `int a=-2,b=1;` Care este valoarea rezultată în urma evaluării expresiei `!(a&&b)==!a||!b;`?

a) `1;`
b) `0;`
c) `TRUE.`

19. Considerăm `int a=0,b=3;` Care este valoarea rezultată în urma evaluării expresiei `!a||++b;` și care este valoarea reținută de `b`?

a) `1` și `3`;
b) `1` și `4`;
c) Aceasta nu este expresie `C++`.

20. Considerăm `int a=-1;` Care este valoarea rezultată în urma evaluării expresiei `-a`?

a) $2^{16}-1$;
b) `0;`
c) -2^{16}.

21. Avem `int a=5,b=3;` Care este valoarea rezultată în urma evaluării expresiei `a&b+a|b+a^b`?

a) 11;
b) nedecidabil;
c) 21.

22. Considerăm `int a=0,b=3;` Care este valoarea rezultată în urma evaluării expresiei `a=b==3+1/3`?

a) 3,333333;
b) 3;
c) 1.

23. Considerăm `int a=0,b=3;` Care este valoarea rezultată în urma evaluării expresiei `a+=b+1`?

a) În partea stângă a unei atribuiri trebuie să figureze o singură variabilă și nu o expresie;
b) 4;
c) 3.

24. Considerăm `int a=0,b=3;` Care este valoarea rezultată în urma evaluării expresiei `a=1=b`?

a) Eroare de sintaxă;
b) 3;
c) 1.

25. Dacă `x` și `y` sunt de tip `int`, ce valori trebuie să rețină pentru ca expresia `x or y=x xor y` să producă valoarea 1?

a) 3 3 b) 3 5 c) 4 7 d) 4 2

26. Dacă `x` este de tip `int`, care dintre expresiile de mai jos produce valoarea 0, indiferent de conținutul variabilei `x`?

a) `x & x` b) `x | x` c) `x ^ x` d) `x | !x`

27. Dacă `x` este de tip `int`, rezolvați ecuația: `(x & x)==x`.

28. Dacă `x` este de tip `int`, rezolvați ecuația: `(x ^ x)==y`

29. Dacă `x` și `y` sunt de tip `int`, rezolvați ecuația: `(x ^ x ^ y)==y`.

30. Dacă `x` este de tip `int`, rezolvați ecuația: `(x | ~x)==x`.

31. Care este funcția evaluată de expresia de mai jos (`x` este real)?
`(x-1)*(x<2)+x*x*(x>=2)`

32. Considerați avantajoasă modalitatea de evaluare a funcțiilor "cu mai multe ramuri" prezentată la problema anterioară?

Răspunsuri:

1. c), d) 2) c) 3. c) 4 a. c) 4 b. c) 4 c. b) 4 d. a) 4 e. c) 4 f. a) 4 g. a) 11. c) 12. b) 13. c) 14. a) 15. b) 16. c) 17. a) 18. a) 19. a) 20. b) 21. a) 22. c) 23. b) 24. a) 25. d) 26. c) 27. Orice număr care poate fi reținut de tipul **int. 28. x=0;**

29. Orice pereche **x y** în care **x** și **y** sunt de tip **int.** 30. **x=-1**

31.
$$f = \begin{cases} x-1, & x < 2 \\ x^2, & \text{altfel} \end{cases}$$

32. Nu, pentru că se efectuează toate comparațiile.

CAPITOLUL 4

Instrucțiunile limbajului C++

4.1. Instrucțiunea `expresie`

Existența acestei instrucțiuni ar putea să ne mire. În pseudocod expresiile intervin în cadrul instrucțiunii de atribuire. În C++ există instrucțiunea expresie. Ea este de forma:

`expresie;`

Ca o curiozitate, limbajul permite să scriem instrucțiunea expresie `7+2;`. Aceasta se evaluează, chiar dacă nu folosim în nici un fel rezultatul. Programul de mai jos este corect!

```
main()
{
 7+2;
}
```

Exemplul dat ilustrează flexibilitatea limbajului. Consecințele acestei instrucțiuni sunt uriașe și vor fi expuse la momentul potrivit. Revedeți expresii C++.

Se observă că instrucțiunea expresie se termină cu `";"`. Dacă expresia este vidă - nu are nici un operand, nici un operator - obținem așa numita instrucțiune vidă. Programul de mai jos este corect!

```
main()
{
 ;
 ;;
}
```

Instrucțiunea expresie este des utilizată în efectuarea atribuirilor (în pseudocod atribuirea este instrucțiune separată).

Exemplul 1. *Programul p1 interschimbă conținutul a două variabile care au fost citite. La sfârșit, se afișează noul conținut al variabilelor.*

```
#include <iostream.h>
main()
{
 int a,b,c;
 cout<<"a=";cin>>a;
 cout<<"b=";cin>>b;
 c=a; a=b; b=c;
 cout<<"a="<<a<<endl;
 cout<<"b="<<b<<endl;
}
```

Exemplul 2. *Se citesc două valori întregi* **a** *și* **b**. *Se cere să se afișeze media aritmetică a lor.*

```
#include <iostream.h>
main()
{
 int a,b;
 float medie;
 cout<<"a=";cin>>a;
 cout<<"b=";cin>>b;
 medie=float(a+b)/2; // De ce am pus float?
 cout<<"media este "<<medie;
}
```

Exemplul 3. *Se citesc 3 numere naturale. Se cere să se afișeze primul număr, suma dintre primul și al doilea, suma primelor 3 numere. Exemplu: dacă se citește 2, 5 și 7, se va tipări 2, 7, 14.*

```
#include <iostream.h>
main()
{
 int s=0,nr;
 cout<<"dati numarul ";cin>>nr;
 s+=nr; //se poate scrie si s=s+nr;
 cout<<s<<endl;
 cout<<"dati numarul ";cin>>nr;
 s+=nr;
 cout<<s<<endl;
 cout<<"dati numarul ";cin>>nr;
 s+=nr;
 cout<<s<<endl;
}
```

4.2. Instrucțiunea `IF`

Ca și în pseudocod, există două forme ale instrucțiunii `if`:

Forma 1.

if (expresie) *instrucțiune*$_1$ else *instrucțiune*$_2$

Principiul de executare este următorul:

⇒ se evaluează expresia;

⇒ dacă valoarea produsă de aceasta este diferită de 0, se execută instrucțiune$_1$;

⇒ dacă valoarea produsă este 0 se execută instrucțiune$_2$.

Forma 2.

if (expresie) *instrucțiune*

Principiul de executare este următorul:

⇒ se evaluează expresia;

⇒ dacă valoarea produsă de aceasta este diferită de **0**, se execută instrucțiunea subordonată.

Exemplul 1. *Programul care urmează calculează maximul dintre două numere citite:*

```
#include <iostream.h>
main()
{
 int a,b,max;
 cout<<"a=";cin>>a;
 cout<<"b=";cin>>b;
 if (a>b) max=a;
 else max=b;
 cout<<"maximul este "<<max;
}
```

Exemplul 2. *Se citesc trei variabile reale* **a, b** *și* **c**. *Să se calculeze*

$$e=\begin{cases} a+b, & c>0 \\ a*b, & c=0 \\ a-b, & c<0. \end{cases}$$

Programul următor rezolvă această problemă, exemplifică folosirea unei instrucțiuni **IF**, plasată în corpul altei instrucțiuni **IF**.

```
#include <iostream.h>
main()
{
 double a,b,c,e;
 cout<<"a=";cin>>a;
 cout<<"b=";cin>>b;
 cout<<"c=";cin>>c;
 if (c>0)e=a+b;
 else
     if (c==0) e=a*b; // se poate si if (!c) ... De ce?
     else e=a-b;
 cout<<e;
}
```

Exemplul 3. *Se citește o valoare întreagă. În cazul în care aceasta este pară (se împarte exact la 2) se va afișa mesajul "am citit un număr par". Altfel, programul nu va da mesaj.*

```
#include <iostream.h>
main()
{
 int nr;
 cin>>nr;
 if (nr%2==0)cout<<"am citit valoare para";
 // se poate si if(!(nr%2)) De ce?
}
```

4.3. Instrucțiunea compusă

Se pune următoarea problemă: cum procedăm în situația în care la utilizarea instrucțiunii **IF** fie într-un caz, fie în altul, urmează să scriem mai multe instrucțiuni. *Pentru a putea scrie mai multe instrucțiuni care să fie interpretate de compilator ca una singură se folosește instrucțiunea compusă.*

Instrucțiunea compusă are forma următoare:

```
{
  i1;
  i2;
  :
  :
  in;
}
```

Aici, $i_1,\ldots,i_n$ reprezintă instrucțiunile care se găsesc în corpul instrucțiunii compuse. Acestea sunt separate prin ';'.

Exemplul 1. *Să se scrie un program care rezolvă ecuația algebrică de gradul 1 cu o necunoscută, cu coeficienți reali (a*x+b=0). Modul de rezolvare a acestei ecuații a fost discutat pe larg atunci când am exemplificat elaborarea algoritmilor prin utilizarea schemelor logice și a limbajului de tip pseudocod. Aici prezentăm programul C++ care rezolvă această problemă. Se observă folosirea unei instrucțiuni compuse la calculul rădăcinii și tipărirea ei.*

```
#include <iostream.h>
main()
{
 float a,b,x;
 cout<<"a=";cin>>a;
 cout<<"b=";cin>>b;
 if (a)
   {
    x=-b/a;
    cout<<x;
   }
 else
  if (b==0) cout<<"infinitatate de solutii";
  else cout<<"nu are solutie";
}
```

Observație: orice program C++ are cel puțin o funcție și aceasta are cel puțin o instrucțiune compusă { }.

4.4 Instrucțiunea switch

Instrucțiunea **switch** are forma generală:

switch (*expresie*)
{
case exp_1: secvență $instrucțiuni_1$; **break;**
case exp_2: secvență $instrucțiuni_2$; **break;**
.....
case exp_n: secvență $instrucțiuni_n$; **break;**
[default: secvență $instrucțiuni_{n+1}$**];**
}

Unde:

- *expresie* are semnificația: expresie de tip întreg;
- exp_i: sunt expresii constante de tip întreg;
- $instrucțiuni_i$ reprezintă o secvență oarecare de instrucțiuni.

Principiul de executare:

- se evaluează expresia;
- dacă aceasta produce o valoare egală cu cea produsă de exp_1, se execută, în ordine, $instrucțiuni_i$ și se trece la instrucțiunea următoare, altfel se execută numai secvența $instrucțiuni_{n+1}$.

Observație 1. Alternativa **default** este facultativă. În absență, în cazul în care nu există coincidență de valori, se trece la instrucțiunea următoare.

Observație 2. Am prezentat o formă simplificată a instrucțiunii **switch**, mai precis, cea corespunzătoare programării structurate. De fapt, **break** este instrucțiune C++, dar utilizarea ei nu este recomandată în perioada de învățare a programării.

Exemplu: Programul care urmează probează instrucțiunea **switch**.

```
#include <iostream.h>
main()
{
  int i;
  cin>>i;
  switch(i)
  {
   case 1:  cout<<"am citit 1";break;
   case 2:  cout<<"am citit 2";break;
   default: cout<<"am citit altceva";
  }
}
```

4.5 Instrucțiunea WHILE

Această instrucțiune reproduce structura de tip `Cât timp ... execută`. Forma generală este:

while (*expresie*) *instrucțiune*

Principiul de executare este următorul:

P1. Se evaluează expresia;

P2. Dacă valoarea produsă de aceasta este diferită de 0, se execută instrucțiunea subordonată, apoi se revine la P1, altfel se trece la instrucțiunea următoare.

Exemplul 1. *Se citește* **n**, *număr natural. Să se calculeze suma cifrelor sale (pentru* **n**=213, *se va tipări 6).*

```
#include <iostream.h>
main()
{
 int n,s=0;
 cout<<"n=";cin>>n;
 while (n)
  {
   s=s+n%10;
   n=n/10;
  }
 cout<<s;
}
```

Exemplul 2. *Se citește* **n**, *număr natural. Să se afișeze numărul obținut prin inversarea cifrelor sale (pentru* **n**=412, *se va tipări 214).*

```
#include <iostream.h>
main()
{
 int n,ninv=0;
 cout<<"n=";cin>>n;
 while (n)
  {
   ninv=ninv*10+n%10;
   n=n/10;
  }
 cout<<"numarul inversat "<<ninv;
}
```

În C++, datorită uriașelor posibilități de lucru cu expresii, de multe ori instrucțiunea subordonată unei instrucțiuni repetitive este cea vidă. În programul de mai jos s-au folosit mai multe expresii separate prin virgulă. Am învățat faptul că valoarea produsă de o astfel de expresie este dată de ultima din șirul lor (cea din dreapta - în exemplu **n**). Nu e mai simplu așa?

```
#include <iostream.h>
main()
{
 int n,ninv=0;
 cout<<"n=";cin>>n;
 while (ninv=ninv*10+n%10,n/=10);
 cout<<ninv;
}
```

Programul mai poate fi scris și așa:

```
#include <iostream.h>
main()
{
 int n,ninv=0;
 cout<<"n=";cin>>n;
 while (ninv*=10,ninv+=n%10,n/=10);
 cout<<ninv;
}
```

4.6 Instrucțiunea DO WHILE

Această instrucțiune este asemănătoare cu structura **Execută...cât timp**. Forma generală a acestei instrucțiuni este următoarea:

do
instrucțiune
while(*expresie*);

Principiul de executare este următorul:

P1. Se execută instrucțiunea subordonată;

P2. Se evaluează expresia. În cazul în care valoarea produsă la evaluare este 0, execuția instrucțiunii **do** se termină, altfel se trece la P1.

Observație: secvența se execută cel puțin o dată, după care se pune problema dacă să se repete sau nu (prin evaluarea expresiei logice).

Exemplul 1. *Se citește un număr natural* **n**, *mai mare sau egal cu 1. Să se calculeze suma primelor* **n** *numere naturale.*

```
#include <iostream.h>
main()
{
 int n,s=0,i=1;
 cout<<"n=";cin>>n;
 do
 {
  s=s+i;
  i=i+1;
 }while (i<=n);
 cout<<s;
}
```

Putem scrie și așa (noi învățăm C++, nu?):

```
#include <iostream.h>
main()
{
 int n,s=0,i=1;
 cout<<"n=";cin>>n;
 do s+=i++;while (i<=n);
 cout<<s;
}
```

Exemplul 2. *Se citește* **n**, *număr natural. Să se descompună în factori primi.*

```
#include <iostream.h>
main()
{
 int n,i=2,fm;
 cout<<"n=";cin>>n;
 do
  {
    fm=0;
    while (n%i==0)
     {
      fm++; // sau fm=fm+1;
      n/=i; // sau n=n/i;
     }
    if (fm) cout<<i<<" la puterea "<<fm<<endl;
    i++;
  }while (n!=1);
}
```

4.7. Instrucțiunea FOR

Instrucțiunea **for** are forma generală:

for(expresie$_{inițializare}$;expresie$_{test}$;expresie$_{incrementare}$) *instrucțiune*

După cum se vede, între paranteze se găsesc 3 expresii.

- Expresie$_{inițializare}$ se folosește, de regulă, pentru inițializarea variabilei de ciclare. Este de remarcat faptul că în cadrul acestei expresii (cu rol special) este posibil chiar să declarăm variabila de ciclare (cu valoare inițială).
- Expresie$_{test}$ se folosește pentru a testa dacă se execută instrucțiunea subordonată - dacă expresia produce la evaluare o valoare diferită de 0, instrucțiunea subordonată for se execută.
- **Expresie$_{incrementare}$** se folosește pentru incrementarea variabilei de ciclare.

Principiul de executare:

P1. Se evaluează $expresie_{inițializare}$ (un caz special este acela în care aceasta conține și declarația variabilei de ciclare);

P2. Se evaluează $expresia_{test}$. În cazul în care aceasta produce o valoare diferită de **0**, se execută instrucțiunea subordonată **for**; apoi se trece la P3, altfel se trece la instrucțiunea următoare (se termină execuția instrucțiunii **for**).

P3. Se evaluează expresia de incrementare și se revine la P2.

Important. Toate expresiile pot fi vide. În concluzie, expresiile de mai sus au rolul precizat în mod normal - dar nu obligatoriu și nici restrictiv. De exemplu, dacă $expresie_{test}$ este vidă, se execută un ciclu infinit.

```
main()
{ for(;;);
}
```

Programul de mai sus ciclează. Dacă un ciclu nu se termină (instrucțiunile se execută la infinit) spunem că programul ciclează. Uneori, din greșeală, scriem programe care ciclează. Pentru a opri din executare un asfel de program, procedăm diferențiat după sistemul de operare în care lucrăm:

- **DOS** - tastăm **CTRL+PAUSE**;
- **WINDOWS** - tastăm **CTRL+ALT+DEL**.

Exemplul 1. Programul *următor* listează numerele **5, 4, 3, 2, 1**.

```
#include <iostream.h>
main()
{ int i;
  for (i=5;i>=1;i--) cout<<i<<" ";
}
```

Exemplul 2. Să se listeze alfabetul în ordine inversă.

```
#include <iostream.h>
main()
{ char car;
  for (car='z';car>='a';car--) cout<<car<<endl;
}
```

Exemplul 3. *Se citește* **n** *(număr natural). Se cere să se efectueze suma primelor* **n** *numere naturale. Exemplu:* **n**=3. **s**=1+2+3=6

```
#include <iostream.h>
main()
{ int i,n,s=0;
  cout<<"n=";cin>>n;
  for(i=1;i<=n;i++)s+=i; //sau s=s+i;
  cout<<"suma primelor n numere naturale este "<<s;
}
```

Se poate și așa...

```
#include <iostream.h>
main()
{
 int i,n,s=0;
 cout<<"n=";cin>>n;
 for(i=1;i<=n;s+=i++);
 cout<<" suma primelor n numere naturale este "<<s;
}
```

Exemplul 4. *Să se calculeze suma: S=0,1+0,2+...+0,9.*

```
#include <iostream.h>
main()
{ int i;
  float s;
  for(i=1;i<=9;i++)s+=(float)i/10;
  cout<<s;
}
```

Având în vedere că variabila de ciclare poate să fie reală, putem scrie și așa:

```
#include <iostream.h>
main()
{ double s=0,i;
  for(i=0.1;i<=0.9;i+=0.1)s=s+i // sau s+=i;
  cout<<s;
}
```

Atenție: Nu este indicat să procedăm astfel. Datorită faptului că numerele reale se reprezintă aproximativ, pot apărea erori. De exemplu, dacă `s` și `i` sunt de tip `float`, rezultatul este eronat. Acesta este motivul pentru care în unele limbaje (de exemplu, Pascal) este obligatoriu ca variabila de ciclare să fie întreagă.

Exemplul 5. *Se citește `n` număr natural. Să se calculeze suma:*

$$S = 1 + 1 \cdot 2 + 1 \cdot 2 \cdot 3 + ... + 1 \cdot 2 \cdot ... \cdot n$$

Ce avem de făcut? Observăm că avem `n` pași. La pasul `i` se adună valoarea `1*2...*i`, iar `i` este cuprins între 1 și `n`.

```
#include <iostream.h>
main()
{ int i,s=0,p=1,n;
  cout<<"n=";cin>>n;
  for(i=1;i<=n;i++)
  { p*=i;//p=p*i;
    s+=p; //s=s+p;
  }
 cout<<s;
}
```

Exemplul 6. *Se citesc* **n** *numere întregi. Se cere să se afișeze cel mai mare număr citit. Exemplu. Dacă avem* **n=4**, *iar numerele sunt -7, 9, 2, 3, se va afișa 9.*

```
#include <iostream.h>
main()
{ int i,max,n,nr;
  cout<<"n=";cin>>n;
  cout<<"nr";cin>>nr;
  max=nr;
  for(i=2;i<=n;i++)
   {
    cout<<"nr";cin>>nr;
    if (nr>max)max=nr;
   }
  cout<<"maximul este "<<max;
}
```

Exercițiu. Refaceți programul și utilizați **do while.**

Exemplul 7. *Se citește un număr natural. Să se afișeze numărul obținut prin inversarea cifrelor sale.*

```
#include <iostream.h>
main()
{ int i,s,n,ninv;
  cout<<"n=";cin>>n;
  for(ninv=0;n>0;)
   { ninv=ninv*10+n%10;
     n=n/10;
   }
 cout<<ninv;
}
```

Se poate si așa. De ce?

```
#include <iostream.h>
main()
{ int i,s,n,ninv;
  cout<<"n=";cin>>n;
  for(ninv=0;n>0;n/=10)ninv=ninv*10+n%10;
  cout<<ninv;
}
```

Cum putem greși?

Analizați programul următor. Credeți că el va tipări numerele între **1** și **5**? Nu. Va tipări **6**. De ce? După **do** am pus caracterul ';' . În acest caz, *instrucțiunea for subordonează instrucțiunea vidă.* Aceasta a fost instrucțiunea care a fost executată în ciclu.

```
#include <iostream.h>
main()
{ int i;
  for(i=1;i<=5;i++); cout<<i<<endl;
}
```

4.8. Ce trebuie să știm pentru a utiliza o funcție ?

Funcțiile sunt apelate cu un număr de parametri efectivi. Parametrii de apel ai unei funcții și tipul ei (natura rezultatului întors) sunt conținuți de prototipul funcției respective. Prototipurile funcțiilor din același domeniu se găsesc grupate într-un fișier **header**. De exemplu, dacă utilizăm funcții de citire/scriere trebuie inclus în textul sursă al programului fișierul **iostream.h**, dacă utilizăm funcții matematice trebuie inclus fișierul **math.h**.

Exemplu de prototip:

```
int t (int, float);
```

Ce informații ne furnizează? În primul rând, funcția se numește **t**. Pentru apelul ei se folosesc 2 parametri: unul de tip **int**, altul de tip **float**. Funcția întoarce un rezultat de tip **int**.

O astfel de funcție poate fi apelată astfel:

- **t(7, 9.2);** primul parametru este **7**, al doilea **9.2**;
- **t(7.3, 9);** dacă parametrii formali nu coincid cu cei efectivi, se efectuează conversia. Astfel, primul parametru ia valoarea **7**, iar al doilea valoarea **9.0**.

Tot așa, funcția poate fi apelată din cadrul unei expresii: **1+t(5,3.2);** În acest caz se evaluează o expresie care rezultă ca sumă între **1** și rezultatul întors de funcție (evident, evaluarea cuprinde și apelul funcției). Atenție: o funcție de tip **void** - care nu returnează rezultatul prin numele ei - nu admite un astfel de apel.

4.9. Funcții "matematice"

Este momentul să prezentăm câteva funcții utile în calcule. Fiecare dintre ele are prototipul în math.h.

➢ Funcția abs are forma generală: **int abs(int x);** Aceasta înseamnă că are un parametru de tip **int**, iar rezultatul este tot de tip **int**. Înaintea numelui se găsește tipul rezultatului. Rolul ei este de a întoarce **|x|** (modulul lui **x**).

Iată cum arată un program care o folosește (tipărește **1234**).

```
#include <iostream.h>
#include <math.h>
main()
{  int n = -1234;
   cout<<abs(n));
}
```

- Funcția **fabs** are forma generală **double fabs(double x);** are același rol cu **abs**, numai că întoarce valoarea unui număr real (chiar **double**).
- Funcția **labs** are forma generală **long int labs(long int x);** are același rol cu abs, numai că întoarce valoarea unui întreg lung.
- Funcția **acos** are forma generală: **double acos(double x);** și calculează valoarea funcției $\arccos(x):[-1,1]\to[0,\pi]$.
- Funcția **asin** are forma generală: **double asin(double x);** și calculează valoarea funcției $\arcsin(x):[-1,1]\to[-\frac{\pi}{2},\frac{\pi}{2}]$.
- Funcția **atan** are forma generală: **double atan(double x);** și calculează valoarea funcției $arctg(x):\Re\to(-\frac{\pi}{2},\frac{\pi}{2})$.
- Funcția **atan2**are forma generală:

  double atan2(double y, double x)

  și calculează $arctg(\frac{y}{x})$. Rezultatul este în intervalul $(-\pi,\pi)$. Motivul?
- Funcția **floor** are forma generală **double floor(double x);** și calculează valoarea rotunjită a lui **x** (rotunjirea se face în minus). Exemple: **floor (123.78)=123, floor (-23,34)=-24.**
- Funcția **ceil** are forma generală **double ceil(double x);** și calculează valoarea rotunjită a lui **x** (rotunjirea se face în plus). Exemple: **ceil(123.78)=124, ceil(-23,34)=-23.**
- Funcția cos are forma generală **double cos(double x);** și calculează valoarea funcției $\cos(x):\Re\to[-1,1]$.
- Funcția **sin** are forma generală **double sin(double x);** și calculează valoarea funcției $\sin(x):\Re\to[-1,1]$.
- Funcția **tan** are forma generală **double tan(double x);** și calculează valoarea funcției $tg(x):\Re-\left\{k\cdot\pi+\frac{\pi}{2}|k\in Z\right\}\to\Re..$
- Funcția **exp** are forma generală **double exp(double x);** și calculează funcția $e^x:\Re\to\Re_+^*$
- Funcția **log** are forma generală **double log(double x);** și calculează funcția $\ln(x):\Re_+^*\to\Re, unde\ \ln(x)=\log_e(x);$

- Funcția **log10** are forma generală **double log10(double x);** și calculează funcția $\lg(x):\Re_+^* \to \Re$, *unde* $\lg(x) = \log_{10}(x)$;
- Funcția **pow** are forma generală **double pow(double x, double y);** și calculează x^y; (iată că în C++ există ridicarea la putere).

Observație. Este sarcina programatorului ca, pentru o astfel de funcție, valoarea argumentului să fie în domeniul de definiție al funcției respective.

4.10. Generarea numerelor aleatoare

Funcțiile prezentate în acest paragraf au prototipurile în **stdlib.h**.

Funcția **rand** are forma:

```
int rand(void);
```

și rolul de a genera un număr aleator întreg cuprins între **0** și **32767**.

Setarea generatorului de numere aleatoare cu un număr aleator se face cu **randomize**.

```
void randomize(void);
```

Exemplu: în programul următor se tipăresc **10** numere aleatoare, cuprinse între **0** și **49**. Observați modul în care am stabilit intervalul în care se găsesc numerele.

```
#include <stdlib.h>
#include <stdlib.h>
#include <iostream.h>
main()
{  int i;
   randomize();
   for(i=0; i<10; i++) cout<<rand() % 50<<' ';
}
```

Pentru setarea generatorului de numere aleatoare într-un anumit punct, se folosește funcția **srand**: **void srand(unsigned n);**

Apelul funcției **srand** cu valoarea **1**, duce la reinițializarea generatorului.

Dacă în programul anterior se înlocuiește **randomize()** cu **srand(n)**, pentru valoarea fixată a lui **n**, seria celor **10** numere va fi întotdeauna aceeași, la rulări succesive ale programului.

4.11. Rularea unei secvențe un interval de timp determinat

În acest paragraf răspundem la întrebarea: *cum putem determina ca o secvență (un program) să ruleze un timp determinat?*

Problema prezintă interes, mai ales în cadrul concursurilor unde există, după cum vom vedea, cazuri în care programele furnizează cea mai "bună" soluție găsită în timpul permis.

În fișierul **time.h** se găsește prototipul funcției **clock**:

```
clock_t clock(void);
```

Tipul funcției este, de fapt, **long** (**clock_t** este o altă denumire pentru acesta, dată cu ajutorul **typedef**). Funcția **clock** întoarce din sistem timpul.

Algoritmul prin care se determină ca o secvență să ruleze un timp determinat este următorul:

- o variabilă reține momentul intrării în secvență;
- se execută secvența;
- o altă variabilă reține timpul;
- secvența este repetată dacă diferența de timp, convertită în secunde, este mai mică decât timpul de lucru permis.

Pentru conversia în secunde, timpul scurs este împărțit la o constantă definită în **time.h** și anume **CLK_TCK**.

Programul următor rulează 30 secunde.

```
#include <time.h>
#include <stdio.h>
int main(void)
{
   long start,end;
   start = clock();
   do
     end = clock();
   while ( (end-start)/CLK_TCK<30);
}
```

Probleme propuse

1. Se citește o valoare întreagă. Să se precizeze dacă este pozitivă sau nu.

2. Se citesc **3** valori reale **a, b, c**. Să se precizeze dacă ele pot fi laturile unui triunghi.

3. Să se scrie un program care rezolvă ecuația $ax^2+bx+c=0$ ($a,b,c \in \Re$).

4. Se citesc **4** numere întregi. Să se afișeze în ordine descrescătoare.

5. Să se calculeze valoarea funcției **F**, definită pe mulțimea numerelor reale, pentru un **x** citit:

$$F=\begin{cases} x*x-3, & x<5 \\ x+1, & 5\le x\le 25 \\ x*x-5*x+6, & x>25 \end{cases}$$

6. Se citesc **a, b, c,** coeficienții unei ecuații de gradul 2. Fără a rezolva ecuația, să se precizeze natura rădăcinilor (reale sau nu, pozitive, negative, semnul rădăcinilor (dacă sunt reale)).

7. Se citesc **n** numere reale. Să se afișeze valoarea minimă citită.

8. Se citește un șir de numere întregi până la întâlnirea numărului **0**. Să se calculeze media aritmetică a numerelor din șir.

9. Citim un șir de numere întregi, la fel ca la problema precedentă. Să se calculeze suma dintre primul număr, al treilea, al cincilea,... și produsul dintre al doilea, al patrulea,... etc.

10. Se citesc **3** numere naturale **n, p, k** și un șir de **n** numere naturale. Câte dintre acestea, împărțite la **p** dau restul **k**?

11. Se citește **n**, număr natural. Să se calculeze expresiile de mai jos (programe separate):

$$E_1 = 1+\frac{1}{2^2}+\frac{1}{3^2}+\ldots+\frac{1}{n^2};$$

$$E_2 = \frac{1}{2\cdot 3}+\frac{2}{3\cdot 4}+\ldots+\frac{n}{(n+1)(n+2)};$$

$$E_3 = 1+\frac{1}{1!}+\frac{1}{2!}+\ldots+\frac{1}{n!},\ unde\ n!=1\cdot 2\cdot\ldots\cdot n.$$

12. Se citește o succesiune de cifre **1** și **0**, prima fiind **1**. Aceasta are semnificația unui număr în baza **2**. Să se afișeze numărul în baza **10**.

13. Se citește o succesiune de caractere (într-o variabilă de tip **char**) **'0', '1', ...,'9', 'a', 'b', ..., 'f'** în care primul caracter nu este 0. Aceasta are semnificația unui număr în baza **16**. Să se afișeze numărul în baza **10**.

14. Se consideră șirul definit astfel (**FIBONACCI**):

$$u(n) = \begin{cases} 1, & dacă\ n = 1\ sau\ n = 2 \\ u(n-1) + u(n-2), & altfel \end{cases}$$

Se citește o valoare naturală **a**. Să se calculeze **u(a)**.

15. Se consideră șirul definit la problema anterioară. Să se afișeze numărul de termeni care sunt strict mai mici decât o valoare citită.

16. Se citesc două numere naturale **m** și **n**. Să se calculeze cel mai mare divizor comun al lor:

⇒ prin utilizarea algoritmului lui EUCLID;
⇒ prin utilizarea relației de mai jos:

$$cmmdc(a,b) = \begin{cases} cmmdc(a-b,b), & dacă\ a>b; \\ cmmdc(a,b-a), & dacă\ a<b; \\ a, & dacă\ a=b. \end{cases}$$

17. Se citește un număr natural. Câte cifre conține?

18. Calculați suma și produsul divizorilor primi ai unui număr citit

19. Se citesc, pe rând, cifrele unui număr (începând cu cifra cea mai semnificativă). Să se reconstituie numărul într-o variabilă de tip **integer**.

20. Se citește un număr natural cu **5** cifre. Afișați numărul format după eliminarea cifrei din mijloc.

21. Se citește un număr cu **8** zecimale (partea întreagă a acestuia este **0**). Să se afișeze numărul rezultat prin eliminarea primelor două și ultimelor două zecimale. Exemplu: **0.12345678**, se va afișa **0.3456**.

22. În practică, un rol fundamental îl joacă numerele aleatoare (întâmplătoare). Există mulți algoritmi care permit generarea lor, însă aceștia implică anumite cunoștințe de matematică ce depășesc cu mult programa de liceu. Aici vom prezenta pe cel mai modest dintre ei, dar și cel mai simplu. Se consideră un număr cu **8** zecimale și se înlătură **4** dintre acestea, întocmai ca la problema anterioară. Numărul obținut se ridică la pătrat. În acest fel obținem un nou număr cu **8** zecimale, cu care se procedează în mod identic. Citiți un număr cu **8** zecimale și afișați primele **n** numere obținute prin procedeul indicat.

Observație: pentru un șir astfel obținut, la un moment dat valorile se repetă (se spune că șirul este periodic). În acest sens, pentru ca șirul să se repete după un număr cât mai mare de valori, nu este indiferent numărul de la care se pleacă. Încercați programul stabilit, pentru mai multe numere de pornire, inclusiv pentru cel definit în problema anterioară.

23. Găsiți un procedeu de generare a unor numere aleatoare întregi care să se găsească între două valori întregi citite.

24. Un număr se numește perfect dacă este egal cu suma divizorilor săi (exclusiv el). Exemplu: **6=1+2+3**. Afișați toate numerele perfecte mai mici decât o valoare dată.

25. Afișați toate numerele naturale mai mici decât o valoare dată, care sunt egale cu suma pătratelor cifrelor lor.

26. Determinați numărul de zile cuprinse între două date calendaristice citite (anii bisecți pe cei divizibili cu 4).

27. Afișați primele **n** perechi de numere prime care sunt consecutive în mulțimea numerelor impare. Exemplu: **(3,5), (5,7)**.

28. Se citește un număr natural par. Să se descompună în sumă de numere prime (conjectura **GOLDBACH**).

29. Se citesc două numere naturale formate din câte 4 cifre distincte. Care este numărul cifrelor comune celor două numere? Cifrele comune nu coincid obligatoriu și ca poziție.

30. Se citesc **2** numere naturale **a** și **n**. Calculați a^n efectuând, pe cât posibil, mai puțin de **n-1** înmulțiri.

Indicație. Vom utiliza formula:

$$a^n = \begin{cases} \left(a^k\right)^2 & , n = 2k; \\ a\left(a^k\right)^2 & , n = 2k+1. \end{cases}$$

pe care o transformăm într-o relație de recurență. La început o scriem așa:

$$a^n = \begin{cases} \left(a^2\right)^k & , n = 2k; \\ a\left(a^{2k}\right) & , n = 2k+1. \end{cases}$$

Notăm $a^n = f(a,n)$. Avem: $f(a,0) = 1$,

$$f(a,n) = \begin{cases} f\left(a^2, \frac{n}{2}\right) & , n = 2k; \\ af\left(a, n-1\right) & , n = 2k+1. \end{cases}$$

Exemplu:

$f(a,10) = f(a^2,5) = a^2 f(a^2,4) = a^2 f(a^4,2) = a^2 f(a^8,1) = a^2 a^8 f(a^8,0) = a^{10}$.

Să observăm că la fiecare iterație se efectuează o înmulțire. Iată câteva rezultate: dacă **n** este **10** se fac **5** înmulțiri, dacă **n** este **25** se fac **7** înmulțiri. E ceva! Dacă am fi procedat în mod clasic, am fi efectuat **24** de

înmulțiri. Numărul înmulțirilor putea să scadă cu **1** dacă nu am fi considerat că îl obținem pe **a** printr-o înmulțire!

Observație foarte importantă. Procedeul de mai sus *nu asigură numărul minim de înmulțiri* pentru calculul lui a^n, chiar dacă așa pare la prima vedere.

31. Se citesc **n** numere naturale. Să se afișeze cel mai mare divizor comun al lor.

32. Se citesc **a**, **b**, **c** coeficienții unei ecuații de gradul **2** și un număr natural **n**. Fără a rezolva ecuația să se calculeze expresia: $S_n=x_1^n+x_2^n$, unde prin x_1 și x_2 am notat rădăcinile ecuației.

Indicație:
x_1 rădăcină => $ax_1^2+bx_1+c=0$ => $ax_1^{n+2}+bx_1^{n+1}+cx_1^n=0$.

Analog, $ax_2^{n+2}+bx_2^{n+1}+cx_2^n=0$.

Prin adunare => $a(x_1^{n+2}+x_2^{n+2})+b(x_1^{n+1}+x_2^{n+1})+c(x_1^n+x_2^n)=0$

=> $aS_{n+2}+bS_{n+1}+cS_n=0$, dar $S_1=-b/a$, $S_2=x_1^2+x_2^2=(x_1+x_2)^2-2x_1x_1=b^2/a^2-2*c/a$.

Programați o astfel de recurență.

33. Rezolvați în mulțimea numerelor întregi ecuația:

$x^2 - y^2 = k$ (**k** natural, citit de la tastatură);

Indicație:
Putem scrie: $(x - y)*(x + y) = k$.; Notăm: $x - y = a$; $x + y = b$;

Rezultă: $x = (a + b) / 2$; $y = (b - a) / 2$.

Pe de altă parte, **a** și **b** trebuie să fie divizori ai lui **k**. De asemenea se ține cont de faptul că, dacă ecuația admite soluția (x,y) admite și soluțiile (–x,–y), (–x,y), (x,–y).

34. Se citesc, pe rând, **n** numere naturale și un număr prim **p**. Se cere să se găsească **k** maxim, astfel încât p^k divide produsul celor **n** numere naturale. Se va evita efectuarea produsului celor **n** numere naturale.

35. Se citesc **k** și **n** numere naturale. Dacă este posibil, să se scrie **n** ca sumă de **k** numere naturale distincte, altfel să se afișeze "IMPOSIBIL".

propusă de autor, Lugoj '97

Indicație. Alegem primele `k-1` numere naturale: `1, 2, 3, ..., k-1`. Notăm cu `s` suma lor. Avem $s=\frac{k(k-1)}{2}$. Dacă `n - s > k - 1`, ultimul număr ales va fi `n - s`. În acest fel ne-am asigurat că am obținut `k` numere distincte. Am folosit faptul că în problemă se cere o soluție, nu interesează care este

aceea. Altfel, se afișează imposibil. Dar de ce afișăm "IMPOSIBIL" în cazul în care în acest fel nu putem obține soluție?

Fie $n_1<n_2<... <n_k$ o altă soluție a problemei: $n_1+ n_2+...+n_k=n$. Dacă $n_1>1$, alegem $n_1=1$, și în locul lui n_2 punem n_2+n_1-1. Dacă $n_2>2$, alegem $n_2=2$, și în locul lui n_3 punem n_3+n_2-2. Procedeul continuă la fel până l-am modificat pe n_{k-1}, iar suma nu a fost afectată.

Exemplu: **n**=10, **k**=3. O soluție este **3, 5, 2.** Aceasta se transformă în:

- **1, 7, 2**
- **1, 2, 7.**

Deci, dacă problema ar fi admis altă soluție, aceasta ar fi putut fi transformată prin procedeul anterior în soluția obținută prin algoritmul prezentat.

36. Suma numerelor. Să se calculeze suma tuturor numerelor formate numai din cifre impare distincte.

Ontario Math, Bull 1979

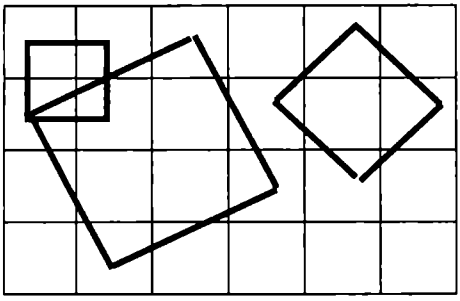
Indicație. Aparent, programul ar trebui să ruleze la infinit. Numai că, un astfel de număr este **13579**. Celelalte, mai mari decât el, se formează prin permutarea cifrelor.

37. Mere și saci. În fiecare din cei **n** saci se găsește același număr de mere. În prima zi se scoate un măr dintr-un sac, în a doua zi se scoate câte un măr din doi saci, ... până când în ultima zi se scot **n** mere din cei **n** saci. Se citește **n**. Se cere, dacă există, o soluție de extragere a merelor din saci, prin care, procedând ca mai sus, toți sacii rămân fără mere. Programul va afișa numărul de mere din fiecare sac, și pentru fiecare zi, din cele **n**, sacii din care se extrag merele.

Parabola, 1978

Indicație. În total se extrag: $1+2+...+n=\frac{n(n+1)}{2}$ mere. Acesta este numărul total de mere care trebuie să existe în saci. Prin urmare, este necesar ca fiecare sac să conțină $\frac{n+1}{2}$ mere. Deci **n** trebuie să fie impar, altfel problema nu are soluție. Dacă **n** este **2m+1**, fiecare sac are **m+1** mere. Ne imaginăm merele așezate în fiecare sac pe verticală. Astfel, totalitatea merelor sunt așezate în **m+1** straturi. Programul va elimina mai întâi merele din primul strat, apoi din al doilea ș.a.m.d.

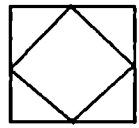
38. Pătrate. Se citesc de la tastatură **m** și **n**, numere naturale strict mai mici decât **50**. Considerăm un dreptunghi cu lățimea **m** și înălțimea **n**, împărțit în **m×n** pătrățele unitare. În cadrul acestuia se pot forma mai multe pătrate care au vârfurile în centrele pătrățelelor unitare, cum sunt cele din figura următoare, unde **m**=6, **n**=4.



Se cere să se afișeze numărul de pătrate care se pot forma.

Problemă propusă de autor, Lugoj '98.

Indicație. Mai întâi vedem câte pătrate distincte cu laturile paralele cu bazele dreptunghiului se pot forma. Avem `(m-1)(n-1)` pătrate cu lungimea laturii `1`. Apoi avem `(m-2)(n-2)` pătrate cu lungimea laturii `2`, `(m-3)(n-3)` pătrate cu lungimea laturii `4` ș.a.m.d. Numărul lor este dat de suma tuturor acestor valori, sumă care merge până la minimul dintre `m` și `n`. Obținem: $\sum_{i=1}^{i=\min(m,n)} (m-i)(n-i)$. Acum: fiecărui pătrat cu lungimea laturii `i`, îi putem asocia `i` pătrate, ca în figura de mai jos:



În acest fel am obținut numărul total de pătrate: $\sum_{i=1}^{i=\min(m,n)} i(m-i)(n-i)$

39. Sportul Regilor Man o' War și Swaps, doi cai minunați din istoria curselor, sunt aduși din nou "să se întreacă". De această dată întrecerea se face pe calculator. Scrieți un program care să simuleze cea mai tare cursă de cai din toate timpurile. Aceasta are următoarele reguli:

a. Traseul este de `1000m`. Câștigătorul este primul cal care parcurge `1000m` sau mai mult.

b. Man o' War avansează `X` metri în timp ce Swaps avansează `Y` metri, după următoarea regulă:

```
X = 9 * Y + 11 - 64 * (INT (( 9 * Y + 11)/64))
Y = 9 * X + 12 - 64 * (INT (( 9 * X + 12)/ 64))
```

c. Pentru a începe cursa, un număr este ales pentru `Y` (citit de la tastatură). Acest "număr rădăcină" este folosit pentru a calcula prima mutare a lui Man o' War. `X` rezultant ajută la primul calcul al lui `Y` (Swaps).

Se cere să se afișeze câștigătorul cursei.

Junior Division 1984

40. Se citesc n numere reale $a_1, \ldots a_n$. Se cere să se calculeze suma:

$$a_1+a_2+\ldots+a_n+\ a_1\ a_2+\ldots+a_{n-1}\ a_n+\ldots+a_1\ a_2\ a_3\ a_4\ldots a_n.$$

Problemă propusă de autor, Sinaia 1998.

Indicație. Se calculează suma coeficienților polinomului și se scade 1:

$$(x+a_1)\ (x+a_2)\ (x+a_3)\ldots\ (x+a_n).$$

Rezultatul este: $(1+\ a_1)\ (1+\ a_2)\ldots\ (1+\ a_n)-1.$

41. Care dintre secvențele următoare afișează corect modulul numărului real memorat de variabila x?

```
a)  cout<<(x<0)*-x+(x>0)*x;
b)  cout<<abs(x);
c)  if (x<0) cout<<-x;
    else cout<<x;
```

42. Care este valoarea pe care o reține variabila rez după executarea secvenței de mai jos?

```
int k,rez;
k=4; rez=3+k/5;
cout<<rez;
```

a) 3;
b) secvența este greșită;
c) 4.

43. Care dintre secvențele de mai jos este corectă din punct de vedere sintactic?

```
k=4;
if (k>=0);
else;
```

```
k=4;
if(k>=0)
 then;
 else;
```

```
k=4;
if k>=0;
else;
```

44. Care dintre secvențele următoare afișează corect prima zecimală a numărului real reținut de x? Exemplu: pentru -2.34 trebuie să se afișeze 3.

```
a) x=fabs(x-floor(x));
   cout<<floor(x*10);
```

```
b) if (x>0) x=x-floor(x);
      else     x=ceil(x)-x;
      cout<<floor(x*10)<<endl;
```

```
c) if (x>0) x=ceil(x)-x;
    else     x=x-floor(x);
   cout<<floor(x*10)<<endl;
```

45. Ce va afișa programul următor:

```
main()
{ int i,s=0;
  for (i=1;i<=10;i++)
  { i=i/10;
    s+=i;}
}
```

a) 0; b) 10 ; c) programul ciclează ; d)1.

46. Care secvență nu este greșită?

a)
```
int x;
...
while (x & x)  x%=2;
cout<<x;
```

b)
```
float x;
...
while (x & x)  x/=2;
cout<<x;
```

c)
```
int x,y;
...
while (x = ++x)
          x=x/2;
 cout<<x;
```

d)
```
int x;
...
while   (x   &   x)
x=2%x;
cout<<x;
```

47. Ce se întâmplă dacă se rulează programul următor?

```
#include <iostream.h>
main()
{ int a=1,b=6;
  while (a!=b)
   { a++; b--;}
  cout<<a;
}
```

a) Programul ciclează;
b) Se afișează 3;
c) Programul conține erori de sintaxă;
d) Nici una din variantele de mai sus nu este adevărată.

48. Ce se întâmplă dacă se rulează programul următor?

```
#include <iostream.h>
main()
{ int x=3;
 while (x!=x*x)
 x=x/2;
 cout<<x;
}
```

a) Programul ciclează;
b) Se afișează 1;
c) Se afișează 0;
d) Nici una dintre variantele de mai sus nu este adevărată.

Testele de la 49 la 52 se referă la programul următor:

```
main()
{ int x=3;
  do
  { x--; x*=2; } while (x!=10);
}
```

49. De câte ori se execută secvența subordonată instrucțiunii do?

a) o dată b) de două ori c) de trei ori d) programul ciclează

50. Înlocuiți cele două atribuiri subordonate instrucțiunii do cu una singură astfel încât programele obținute să fie echivalente.

51. Dacă valoarea inițială a lui x este 5 (x=5), care trebuie să fie condiția de continuare (while (x=?)) astfel încât atribuirile subordonate lui do să se execute exact de 4 ori?

a) 10 b) 8 c) 50 d) 28

52. Pentru care dintre valorile inițiale ale lui x (de mai jos) programul nu ciclează?

a) 0 b) 1 c) 2 d) 4

53. Care dintre atribuirile de mai jos înlocuiește secvența celor trei atribuiri: x:=x-1; x:=2*x; x:=x+1;

a) x:=2*x-1 b) x:=2*x-3 c) x:=2*x+1 d) x:=2*x

Răspunsuri:

41. a) c) 42. a) 43. a) 44. b) 45. c) 46. d) 47. a) 48. b) 49. c) 50. x=2*(x-1); 51. c) 52. d) 53. a)

CAPITOLUL 5

Tablouri

5.1 Tabloul în interpretare matematică

Fie $A_{n_i} = \{1,2,...,n_i\}$ mulțimea primelor n_i numere naturale. Fie $M = A_{n_1} \times A_{n_2} \times ..... \times A_{n_k}$ produsul cartezian a **k** astfel de mulțimi.

Se numește tablou o funcție $f: M \to T$, unde T este o mulțime oarecare. Numărul **k** este dimensiunea tabloului. Dacă **k**=1 tabloul se mai numește și vector. Vectorul are n_1 componente. Dacă **k**=2 tabloul se mai numește și matrice. Matricea are $n_1 \times n_2$ elemente.

Exemple:

1. Vector cu **5** componente numere naturale

$$V = (1,3,6,2,4)$$

Aici **k**=1; n_1=5, **T=N**, elementele vectorului sunt v_1=1, v_2=3,...,v_5=4

2. Vector cu **n** componente numere reale

$$Z = (z_1, z_2, z_3, ..., z_n), \quad z_i \in \Re, i \in \{1,2,...,n\}$$

Aici **k**=1, n_1=**n**, **T**=$\Re$.

3. Matrice cu **n** linii și **m** coloane, cu elemente din $\Re$:

$$A = \begin{pmatrix} a_{1,1} & a_{1,2} & & a_{1,n} \\ a_{2,1} & a_{2,2} & & . \\ a_{3,1} & a_{3,2} & & a_{3,n} \\ . & & & . \\ & & & \\ a_{m-1,1} & a_{m-1,2} & . \ . & a_{m-1,n} \\ a_{m,1} & a_{m,2} & & a_{m,n} \end{pmatrix} \quad a_{i,j} \in \Re$$

⇒ Tabloul de mai sus se numește A;
⇒ Elementele sale sunt $a_{1,1}, a_{1,2}, \ldots, a_{m,n}$.
⇒ Prin $a_{i,j}$ se înțelege elementul care se găsește în linia **i** și coloana **j**.

Aici **k**=2, n_1=**m**, n_2=**n**; **T**=$\Re$.

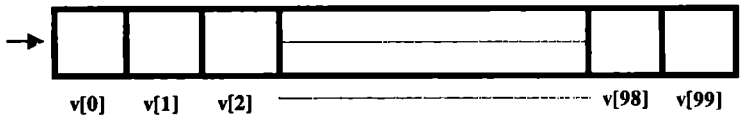
Atenție! Frecvent se confundă numărul componentelor cu dimensiunea tabloului. De exemplu, despre un vector cu **n** componente se spune că este de dimensiune **n** (când, de fapt, toți vectorii au dimensiunea 1).

5.2 Tablouri în C++

În C++ tipul tablou se declară astfel:

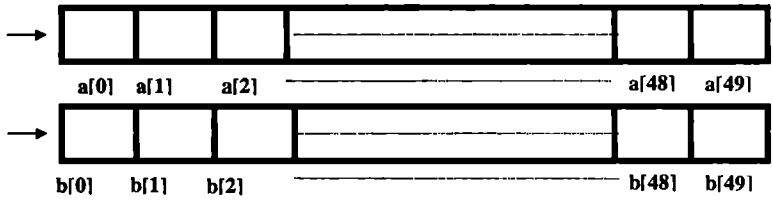
***tip* nume[numar natural$_1$][numar natural$_2$]...[numar natural$_n$]**

Exemplul 1. int `v[100];`



Am declarat un vector cu `100` de componente de tip întreg. *Adresarea unei componente se face prin indice*, care se trece între paranteze drepte. Atenție! *Componentele au indicii între* `0` și `99`. Această regulă este valabilă în general. De exemplu, dacă dorim să adresăm componenta a doua, scriem `v[1]`.

Exemplul 2. `float a[50], b[50];` Am declarat doi vectori, `a` și `b` care au componente de tip `float`.



Exemplul 3. `int a[10][9];`

Am declarat o matrice (tablou) `a` care are `10` linii și `9` coloane. Liniile sunt de la `0` la `9`, iar coloanele de la `0` la `8`. Fiecare element al ei este de tip `int`. Ele se adresează astfel `a[0][0], a[0][1],...a[9][8]`.

Limbajul `C++` nu ne permite să declarăm o variabilă tablou cu număr variabil de componente (există și limbaje care permit aceasta, de exemplu, *Basic*). De multe ori nu știm câte componente vor fi necesare pentru o anumită rulare a programului. Orice problemă în care se lucrează cu variabile de tip tablou și în care se cere prelucrarea unui număr (care nu se cunoaște de la început) de componente, constituie un exemplu în acest sens. Atunci ce facem?

Ideea este să *rezervăm un număr maxim de componente - atât cât este necesar pentru rulare atunci când* `n` *este maxim. La fiecare rulare a programului se cere numărul de componente. De cele mai multe ori o parte dintre ele rămân neutilizate.*

Exemplul 1. Să analizăm programul următor. Acesta citește și tipărește variabila din exemplul 1. Inițial, se citește `n` (numărul componentelor). După aceasta se citesc, pe rând, toate componentele. Să presupunem că s-a citit

n=3. Apoi se citesc 3 numere întregi, câte unul pentru fiecare componentă, și anume **6**, **3**, **4**. Citirea s-a făcut cu ajutorul instrucțiunii **for**, unde variabila de ciclare reprezintă chiar indicele componentei care se citește (**i** ia valori între **0** și **2**). În final, se tipărește conținutul componentelor citite. Restul componentelor vor reține valoarea inițială (reziduală, pentru că aceasta nu este cunoscută).

```
#include <iostream.h>
main()
{
  int v[100],n,i;
  cout<<"numar de componente ";cin>>n;
  for (i=0;i<n;i++)
   {
    cout<<"v["<<i+1<<"]=";
    cin>>v[i];
   }
  for (i=0;i<n;i++)cout<<v[i]<<endl;
}
```

Înainte de a citi o componentă afișăm numele variabilei, urmat de indicele componentei (între paranteze) și semnul '='. De exemplu, se va afișa:

- **v[1]=_** și se așteaptă introducerea valorii pentru prima componentă;
- **v[2]=_** și se așteaptă introducerea valorii pentru a doua componentă;
- **v[3]=_** și se așteaptă introducerea valorii pentru a treia componentă.

Observați faptul că "utilizatorul" a fost scutit de amănuntele limbajului - indicele afișat de program este între **1** și **3**, așa cum presupunem că se așteaptă...

Observație: *Nu sunt permise atribuiri de forma b=a (unde a și b sunt tablouri). În acest caz atribuirea se face pe componente.*

Exemplul 2. Programul de mai jos utilizează doi vectori, cu componente de tip **float**. Se citește vectorul **a**. Apoi se face atribuirea de mai sus, pe componente. La sfârșit se tipărește vectorul **b**.

```
#include <iostream.h>
main()
{
  int n,i;
  float a[50], b[50];
  cout<<"numar de componente ";cin>>n;
  for (i=0;i<n;i++)
   {
    cout<<"a["<<i+1<<"]=";
    cin>>a[i];
   }
  for (i=0;i<n;i++)b[i]=a[i];
  for (i=0;i<n;i++)cout<<b[i]<<endl;
}
```

Exemplul 3. În programul următor se citește și tipărește un tablou. Inițial se citesc numărul de linii și coloane ale tabloului (**m** și **n**). Observați modul în care am afișat tabloul - de așa natură încât și pe ecran să arate ca un tablou.

```
#include <iostream.h>
main()
{
  int m,n,i,j,a[9][9];
  cout<<"m=";cin>>m;
  cout<<"n=";cin>>n;
  for (i=0;i<m;i++)
   for(j=0;j<n;j++)
   {
    cout<<"a["<<i+1<<','<<j+1<<"]=";
    cin>>a[i][j];
   }
  for (i=0;i<m;i++)
  {
   for (j=0;j<n;j++)cout<<a[i][j]<<' ';
   cout<<endl;
  }
}
```

În memorie, tablourile sunt memorate pe linii. Aceasta înseamnă că la început este memorată prima linie, apoi a doua s.a.m.d.

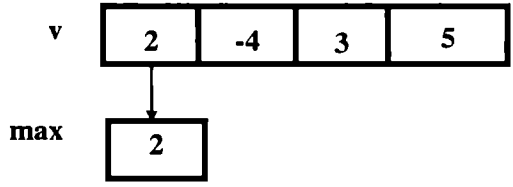
5.3. Algoritmi fundamentali care lucrează cu vectori

În acest paragraf vom prezenta câțiva algoritmi clasici, fundamentali în informatică.

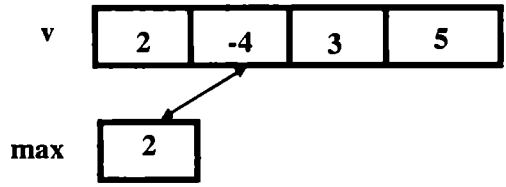
5.3.1. Maxim, minim

Se citește un vector cu n componente numere întregi. Se cere să se afișeze cel mai mare număr întreg găsit. Exemplu: **n=4** și se citesc valorile **2**, **-4**, **3**, **5**. Se va afișa **5**.

O variabilă (**max**) va prelua conținutul primei componente. Pe rând, conținutul acesteia va fi comparat cu numerele întregi reține de componentele **2, 3,...,n**. În cazul în care se găsește un număr mai mare decât cel reținut de variabila **max**, aceasta va reține acel număr. Iată situația inițială:



-4<2 (conținutul variabilei **max**). Se trece mai departe.



În final, valoarea maximă găsită este **5**.

```
#include <iostream.h>
int v[9],n,i,max;
main()
{ cout<<"n=";cin>>n;
   for(i=0;i<n;i++)
    { cout<<"v["<<i+1<<"]=";cin>>v[i];
    }
   max=v[0];
   for(i=1;i<n;i++)
     if (v[i]>max)max=v[i];
   cout <<"valoarea maxima citita este "<<max;
}
```

Exercițiu: Care este modificarea ce trebuie efectuată în program pentru ca acesta să afișeze valoarea minimă?

5.3.2. Elemente distincte

Se citește n și o variabilă de tip array cu n componente numere întregi. Să se decidă dacă numerele citite sunt distincte (nu există două numere egale).

Pentru a rezolva problema trebuie testate toate perechile de numere întregi. Cum putem face aceasta? Ideea este următoarea:

- se compară valoarea reținută de prima componentă cu valorile reținute de componentele de la **2** la **n**;
- se compară valoarea reținută de a doua componentă cu valorile reținute de componentele de la **3** la **n**;
- . . .
- se compară valoarea reținută de componenta **n-1** cu valoarea reținută de componenta **n**.

Dacă în urma comparațiilor nu se găsesc două valori egale, rezultă că numerele citite sunt distincte, altfel acestea nu sunt distincte. Exemplu: se citesc **n=6** și valorile **1**, **3**, **2**, **4**, **3**, **5**.

O variabilă **i** reține indicele primei componente supusă comparației și o altă variabilă **j** reține indicele celei de-a doua componente.

Când **i** ia valoarea **1**, **j** va lua, pe rând, valorile **2**, **3**,...,**n**.
Când **i** ia valoarea **2**, **j** va lua, pe rând, valorile **3**, **4**,...,**n**.
...
Când **i** ia valoarea **n-1**, **j** va lua valoarea **n**.

În concluzie, pentru fiecare **i** între **1** și **n-1**, **j** va lua valorile cuprinse între **i+1** și **n** (aceasta justifică existența celor două cicluri for). Unei variabile "logice" **gasit**, cu valoarea inițială **0**, i se atribuie valoarea **1** dacă sunt găsite două valori egale. Testul se termină fie când au fost comparate toate valorile, fie când au fost găsite două valori egale. Vedeți modul în care am scris condițiile de continuare ale celor două cicluri **for**. Este un exemplu care demonstrează flexibilitatea limbajului **C++**. În final, se tipărește răspunsul în funcție de valoarea reținută de variabila **gasit**.

```
#include <iostream.h>
int v[9],n,i,j,gasit;
main()
{
 cout<<"n=";cin>>n;
 for(i=0;i<n;i++)
 {
  cout<<"v["<<i+1<<"]=";cin>>v[i];
 }
 gasit=0;
 for(i=0;i<n && !gasit;i++)
  for (j=i+1;j<n && !gasit;j++)
     if (v[i]==v[j])gasit=1;
 if (gasit) cout <<"numerele nu sunt distincte";
 else cout<<"numerele sunt disticte";
}
```

5.3.3. Mulțimi

Mulțimile au fost studiate la matematică. Am învățat faptul că ele *nu se pot defini,* dar pot fi descrise. De exemplu, descriem noțiunea de mulțime ca fiind dată de mai multe elemente de același tip. În cadrul unei mulțimi un element apare o singură dată. De exemplu, **{3, 5, 3, 2}** nu este mulțime pentru că elementul **3** apare de două ori.

În acest paragraf prezentăm programe care realizează principalele operații cu mulțimi (reuniune, intersecție, etc).

A citi o mulțime, înseamnă a introduce elementele care o alcătuiesc. Acestea sunt memorate într-o variabilă de tip *vector. Presupunem că elementele introduse sunt distincte.*

a. Testul de apartenență. *Se citește o mulțime A de numere întregi. Se citește un număr întreg e. Să se decidă dacă* $e \in A$.

Pentru rezolvare, procedăm într-un mod clasic. O variabilă **gasit** va reține, inițial, valoarea **0**. Apoi se testează fiecare element al mulțimii **A** dacă este sau nu egal cu numărul reținut de **e**. În caz de egalitate variabila **gasit** va reține **1**. La sfârșit, se va da răspunsul în funcție de conținutul variabilei **gasit**.

```
#include <iostream.h>
int mult[9],n,e,i,gasit;
main()
{ cout<<"numarul de elemente ale multimii ";cin>>n;
  for(i=0;i<n;i++)
   { cout<<"mult["<<i+1<<"]=";cin>>mult[i];
   }
  cout<<"elementul considerat ";cin>>e;
  gasit=0;
  for(i=0;i<n && !gasit;i++)
     if (mult[i]==e)gasit=1;
  if (gasit) cout <<"elementul apartine multimii";
  else cout<<"elementul nu apartine multimii";
}
```

b. Diferența a două mulțimi. *Se citesc două mulțimi A și B. Se cere să se afișeze mulțimea C=A-B.*

Cunoaștem modul în care se definește diferența a două mulțimi:

$$C = A - B = \{x \in A | x \notin B\},$$

adică elemente care aparțin mulțimii A și nu aparțin mulțimii B.

De aici rezultă algoritmul: *pentru fiecare element al mulțimii A, se face testul dacă aparține sau nu mulțimii B. În caz de neapartenență, acesta este adăugat unei mulțimi C, inițial vide.* O variabilă `k` (cu valoarea inițială `0`) reține indicele componentei din `c` care va memora următorul element ce se adaugă mulțimii diferență. În final, se tipărește `c`. De ce afișarea se face numai pentru componentele de indice între `0` și `k-2`?

```
#include <iostream.h>
int multa[9],multb[9],multc[9],n,m,i,j,k,gasit;
main()
{
 cout<<"numarul de elemente al multimii A ";cin>>n;
 for(i=0;i<n;i++)
 { cout<<"mult["<<i+1<<"]=";cin>>multa[i];
 }
 cout<<"numarul de elemente al multimii B ";cin>>m;
 for(i=0;i<m;i++)
 { cout<<"mult["<<i+1<<"]=";cin>>multb[i];
 }
 k=0;
 for (i=0;i<n;i++)
  { gasit=0;
    for (j=0;j<=m && !gasit;j++)
     if (multa[i]==multb[j])gasit=1;
   if(!gasit)multc[k++]=multa[i];
  }
 cout<<"A-B"<<endl;
 for(i=0;i<k;i++)cout<<multc[i]<<endl;
}
```

c. Reuniunea a două mulțimi. *Se citesc două mulțimi de numere întregi A și B. Se cere să se afișeze mulțimea C=A∪B.*

Pentru rezolvare, se rezervă trei variabile de tip *tablou* cu componente care reține numere întregi (**A**, **B**, **C**). După citirea celor două mulțimi **A** și **B**, se listează mulțimea **B**, apoi **A-B**. De fapt, aplicăm formula:

$$C=A\cup B=B\cup(A-B).$$

```
#include <iostream.h>
int multa[9],multb[9],multc[9],n,m,i,j,k,gasit;
main()
{
 cout<<"numarul de elemente al multimii A ";cin>>n;
 for(i=0;i<n;i++)
 {
  cout<<"mult["<<i+1<<"]=";cin>>multa[i];
 }
  cout<<"numarul de elemente al multimii B ";cin>>m;
 for(i=0;i<m;i++)
 {
  cout<<"mult["<<i+1<<"]=";cin>>multb[i];
 }
 k=0;
 for (i=0;i<n;i++)
  {
   gasit=0;
   for (j=0;j<=m && !gasit;j++)
    if (multa[i]==multb[j])gasit=1;
   if(!gasit)multc[k++]=multa[i];
  }
 cout<<"A reunit cu B"<<endl;
 for(i=0;i<m;i++)cout<<multb[i]<<endl;
 for(i=0;i<k;i++)cout<<multc[i]<<endl;
}
```

d. Intersecția a două mulțimi. *Se citesc două mulțimi de numere întregi A și B. Se cere să se afișeze mulțimea C=A∩B.*

Se cunoaște definiția intersecției a două mulțimi:

$$C = A \cap B = \{x \in A | x \in B\}$$

Pornind de la definiție, se deduce imediat algoritmul: *pentru fiecare element al mulțimii A se face testul de apartenență la mulțimea B, iar în caz afirmativ, este adăugat la o mulțime C, inițial vidă.*

```
#include <iostream.h>
int multa[9],multb[9],multc[9],n,m,i,j,k,gasit;
main()
{
 cout<<"numarul de elemente al multimii A ";cin>>n;
 for(i=0;i<n;i++)
  { cout<<"mult["<<i+1<<"]=";cin>>multa[i];
  }
```

```
cout<<"numarul de elemente al multimii B ";cin>>m;
for(i=0;i<m;i++)
{
 cout<<"mult["<<i+1<<"]=";cin>>multb[i];
}
k=0;
for (i=0;i<n;i++)
 {
  gasit=0;
  for (j=0;j<=m && !gasit;j++)
   if (multa[i]==multb[j])gasit=1;
  if(gasit)multc[k++]=multa[i];
 }
cout<<"A intersctat cu B"<<endl;
for(i=0;i<k;i++)cout<<multc[i]<<endl;
}
```

e. Produsul cartezian dintre două mulțimi. *Fie mulțimile A={1,2...,n} și B={1,2...,m} (m și n se citesc). Se cere să se afișeze mulțimea C=A×B.*

Cunoaștem relația: $C = A \times B = \{(x, y) | x \in A, y \in B\}$. Exemplu: A={1,2}, B={1,2,3}, C=A×B={(1,1), (1,2), (1,3), (2,1), (2,2) (2,3)}. Dacă urmărim cu atenție, observăm că trebuie afișate toate perechile de numere (**x,y**) cu **x**∈{1,...,**n**} și **y**∈{1,...,**m**}. Aceasta se realizează foarte ușor cu două cicluri **for** imbricate.

```
#include <iostream.h>
main()
{
 int n,m,i,j;
 cout<<"numarul de elemente al multimii A ";cin>>n;
 cout<<"numarul de elemente al multimii B ";cin>>m;
 for(i=1;i<=n;i++)
  for (j=1;j<=m;j++)
   cout<<i<<" "<<j<<endl;
}
```

După cum observați, am rezolvat problema pentru un caz particular (mulțimea **A** este formată din numerele naturale cuprinse între **1** și **n**, iar mulțimea **B** este formată din numerele naturale cuprinse între **1** și **m**).

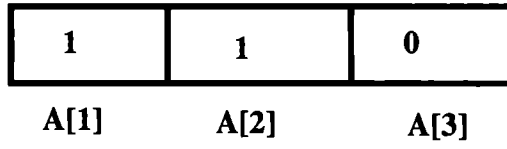
Dar, în cazul general, care este algoritmul? De exemplu, în cazul în care *A este o mulțime formată din n caractere, iar B este o mulțime formată din m caractere*. Problema este interesantă, se poate aplica și pentru alți algoritmi, și admite o rezolvare neașteptat de simplă. În primul rând, se citesc elementele celor două mulțimi. Exemplu: **A**={a, b} și **B**={c, d, e}. Mulțimea **A** are două elemente, iar mulțimea **B** are 3. Rezolvăm problema ca în cazul anterior, pentru mulțimile {1, 2} și {1, 2, 3}. Diferența este că, la afișare, *în loc să afișăm perechea (i, j), afișăm perechea (A[i], B[j])*. Astfel, în loc să afișăm (1,1), afișăm (**A[1], B[1]**) adică (a, c) ș.a.m.d.

```
#include <iostream.h>
char multa[9],multb[9];
int n,m,i,j;
main()
{
 cout<<"numarul de elemente al multimii A ";cin>>n;
 for(i=0;i<n;i++)
 {
  cout<<"mult["<<i+1<<"]=";cin>>multa[i];
 }
  cout<<"numarul de elemente al multimii B ";cin>>m;
 for(i=0;i<m;i++)
 {
  cout<<"mult["<<i+1<<"]=";cin>>multb[i];
 }

 for (i=0;i<n;i++)
  for (j=0;j<m;j++)
   cout<<multa[i]<<"  "<<multb[j]<<endl;
}
```

f. Submulțimi. *Se citește n, număr natural. Se cere să se afișeze toate submulțimile mulțimii {1,2,...,n}.*

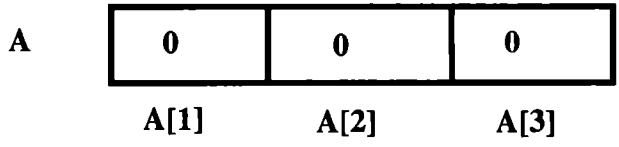
Exemplu `n`=3. Mulțimea {1, 2, 3} are următoarele submulțimi: {1, 2, 3}, {1, 2}, {1, 3}, {2, 3}, {1}, {2}, {3} și ∅ (mulțimea vidă).

Pentru rezolvarea problemei trebuie *să lămurim modul în care se poate memora o submulțime a mulțimii {1, 2,...,n}.* O submulțime se poate memora sub forma unei variabile de tip *tablou* cu `n` componente, unde fiecare componentă reține 0 sau 1. Componenta `i` ia valoarea `1` dacă elementul `i` aparține submulțimii și `0` în caz contrar (o astfel de reprezentare se numește reprezentare prin vector caracteristic). Exemplu: Fie submulțimea `{1, 2}` a mulțimii `{1, 2, 3}`.

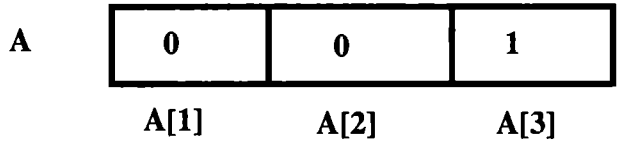


Avem `A[1]=1`, pentru că elementul `1` aparține submulțimii considerate, `A[2]=1`, pentru că elementul `2` aparține submulțimii și `A[3]=0`, pentru că elementul `3` nu aparține submulțimii. Și acum, atenție: ce înseamnă generarea tuturor submulțimilor? Înseamnă generarea tuturor combinațiilor de `0` și `1` care pot fi reținute de vectorul `A`. Dar o astfel de combinație poate fi interpretată ca un număr natural scris în binar. Atunci? Trebuie să generăm toate numerele (în binar) care se pot reprezenta utilizând `n` cifre. Ce presupune aceasta? Pornind de la `00...000`, se adună la fiecare pas `1`, simulând adunarea în binar. Astfel, obținem o nouă combinație de `0` și `1` care reprezintă numărul din pasul precedent la care s-a adunat o unitate.

Când ne oprim? Atunci când a fost scrisă ultima combinație **11...11**, care corespunde celui mai mare număr care poate fi reținut în cele **n** componente. Cum, testăm aceasta? Simplu, la fiecare pas se calculează suma numerelor reținute de cele **n** componente. În momentul în care aceasta este **n**, înseamnă că au fost generate toate numerele. Exemplu: Se citește **n=3** și se dorește afișarea tuturor submulțimilor mulțimii **{1, 2, 3}**. Se pornește de la configurația:

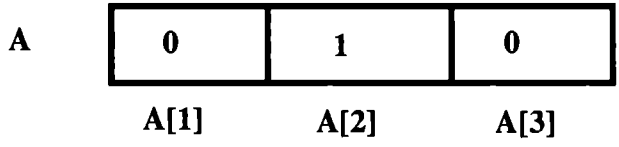


Se adună **1** și se obține:

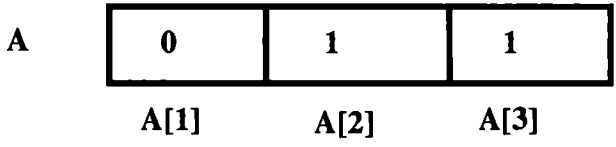


Această configurație reprezintă numărul **1** precum și submulțimea {3}.

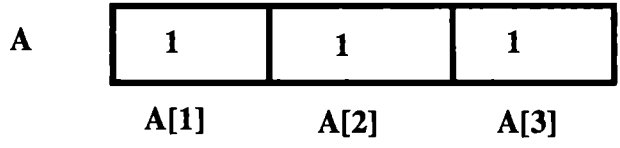
Se adună 1:



Configurația reprezintă numărul **2** precum și submulțimea {2}. Adunăm 1 și obținem:



Procedeul continuă până se obține configurația:



Aceasta reprezintă submulțimea {1, 2, 3} (una din submulțimile unei mulțimi este chiar mulțimea propriu-zisă). Dacă ne gândim la modul în care

am generat submulțimile unei mulțimi cu **n** elemente, ajungem la concluzia că există 2^n submulțimi ale ei (inclusiv mulțimea vidă). Iată și programul:

```
#include <iostream.h>
int multa[9],n,i,s;
main()
{
 cout<<"numarul de elemente al multimii A ";cin>>n;
 for(i=0;i<n;)multa[i++]=0; // se poate si asa
 do
 {
  multa[n-1]++;
  for (i=n-1;i>=1;i--)
   if (multa[i]>1)
    {
     multa[i]-=2;
     multa[i-1]+=1;
    }
   s=0;
   for(i=0;i<n;i++)s+=multa[i];
   for(i=0;i<n;i++)
    if(multa[i]) cout<<i+1<<' ';
    cout<<endl;
 } while(s<n);
 cout<<"multimea vida ";
}
```

5.3.4 Metode de sortare

Sortare. *Se citesc n numere întregi. Se cere ca acestea să fie scrise în ordine crescătoare (descrescătoare).* Exemplu. Se citește **n=4** și se citesc numerele: **3, 1, 6, 2**. Numerele vor fi afișate în ordinea **1, 2, 3, 6** (crescătoare) sau **6, 3, 2, 1** (descrescătoare).

Operația prin care se aranjează cele n numere în ordine crescătoare (descrescătoare) se numește sortare.

Pentru a sorta cele **n** valori, acestea se citesc într-o variabilă de tip tablou. Sortarea propriu-zisă se face în cadrul acestei variabile. Se cunosc o mulțime de algoritmi de sortare. Aceștia au fost elaborați în timp. Noi vom învăța câțiva dintre ei, cei mai ușor de înțeles. Aceasta nu înseamnă că sunt și cei mai eficienți. Dimpotrivă, nu excelează din acest punct de vedere.

Înainte de a trece la prezentarea lor, răspundem la întrebarea: *care este natura valorilor care se pot sorta?* Răspunsul este simplu: *orice valoare care aparține unui tip asupra căruia pot acționa operatorii relaționali* (<, >, <=, >=).

Prin urmare:

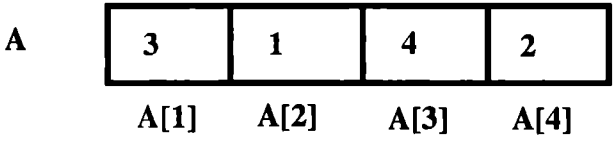
- Valorile care aparțin unuia dintre tipurile întregi pot fi sortate;
- Valorile aparținând unuia din tipurile reale pot fi sortate.

În continuare, prezentăm câțiva algoritmi de sortare. Pentru exemplificare, se consideră *că se citesc n numere întregi care trebuie sortate crescător.* Fiecare din programe, poate fi modificat cu ușurință pentru sortarea altor valori permise. Vom presupune că se citesc **4** valori **3, 1, 4, 2.**

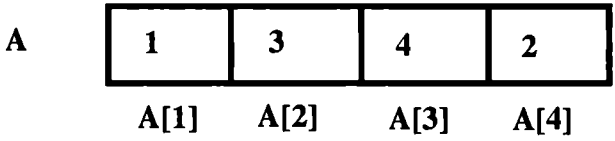
a. Sortarea prin selectarea minimului (maximului). Algoritmul este:

- *se determină minimul dintre valorile reținute, începând cu prima poziție;*
- *minimul este trecut în prima poziție, prin interschimbarea conținuturilor dintre componenta de indice 1 și componenta care îl memorează;*
- *se determină minimul dintre valorile reținute, începând cu a doua poziție;*
- *minimul este trecut în poziția doi, prin interschimbarea conținuturilor dintre componenta de indice 2 și componenta care îl memorează;*
- *se determină minimul dintre valorile reținute, începând cu a treia poziție;*
- *minimul este trecut în poziția 3, prin interschimbarea conținuturilor dintre componenta de indice 3 și componenta care îl memorează;*
- . . .
- *se determină minimul dintre valorile reținute, începând cu penultima poziție;*
- *minimul este trecut în penultima poziție, prin interschimbarea conținuturilor dintre componenta de indice n-1 și componenta care îl memorează.*

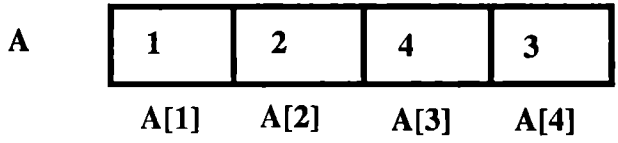
Exemplu. Conținutul variabilei care reține numerele citite este:



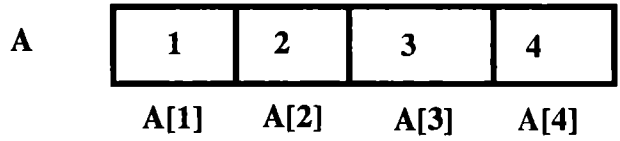
Minimul este **1** și se găsește în poziția **2**. Se inversează conținuturile componentelor **1** și **2**:



Se determină minimul dintre valorile reținute în componentele de la **2** la **4**. Acesta este **2** și este reținut de componenta **4**. Se inversează conținutul componentelor **2** și **4**.



Se determină minimul dintre valorile reținute în componentele de la 3 la 4. Acesta este 3 și este reținut de componenta 4. Se inversează conținutul componentelor 3 și 4.



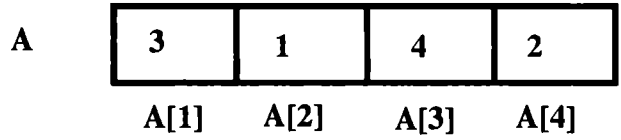
Acum valorile se pot afișa. Evident, procedeul a fost repetat de n-1 ori. În cazul în care se cere sortarea descrescătoare, la fiecare pas se calculează maximul (exercițiu).

```
#include <iostream.h>
int a[9],n,i,j,k,man,min;
main()
{
 cout<<"n=";cin>>n;
 for(i=0;i<n;i++)
 {
  cout<<"a["<<i+1<<"]=";cin>>a[i];
 };
 for(i=0;i<n-1;i++)
 {
  min=a[i];k=i;
  for (j=i+1;j<n;j++)
   if (a[j]<min)
    {
     min=a[j];
     k=j;
    }
   man=a[k];
   a[k]=a[i];
   a[i]=man;
 }
 for(i=0;i<n;i++) cout<<a[i]<<endl;
}
```

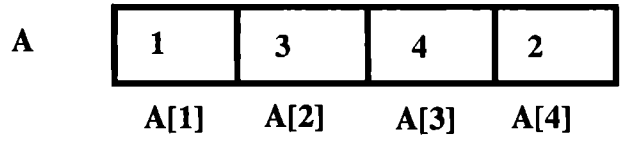
b. Sortarea prin interschimbare. Algoritmul este următorul:

- *se parcurge variabila inversând conținuturile componentelor alăturate care nu sunt în ordine crescătoare (descrescătoare);*
- . . .
- *procedeul se repetă până când are loc o parcurgere în care nu se fac inversări.*

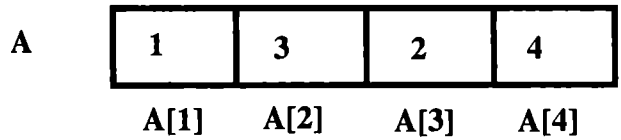
Fie situația inițială:



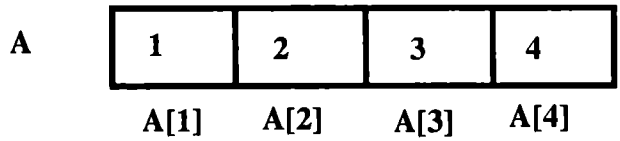
Se efectuează prima parcurgere. Se inversează valorile reținute de componentele **1** și **2**. Se obține:



Valorile reținute de componentele **2** și **3** nu se inversează. În schimb, se inversează valorile reținute de componentele **3** și **4**.



Întrucât au avut loc inversări se reparcurge vectorul. Se inversează valorile reținute de componentele **2** și **3**:



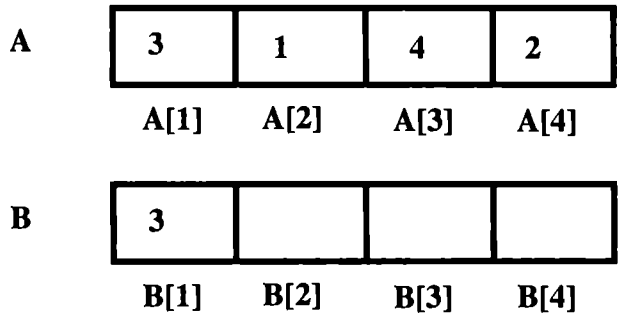
Alte inversări nu se fac. Întrucât în actuala parcurgere au fost făcute inversări, se reparcurge vectorul, de data aceasta inutil și algoritmul se încheie.

```
#include <iostream.h>
int a[9],n,i,k,man,gasit;
main()
{
 cout<<"n=";cin>>n;
 for(i=0;i<n;i++)
 {
  cout<<"a["<<i+1<<"]=";cin>>a[i];
 };
 do
 {  gasit=0;
   for(i=0;i<n-1;i++)
   if (a[i]>a[i+1])
   { man=a[i]; a[i]=a[i+1]; a[i+1]=man;
     gasit=1;
   }
 }while (gasit);
 for(i=0;i<n;i++) cout<<a[i]<<endl;
}
```

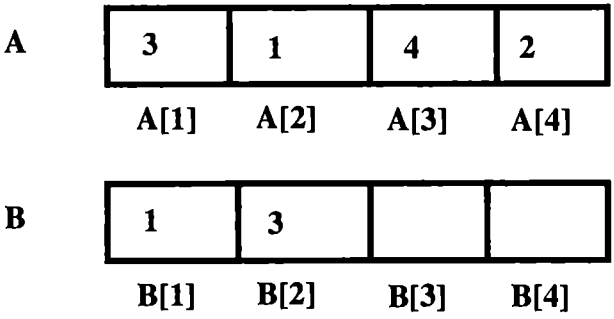
c. Sortarea prin inserție. Cu ajutorul variabilei citite `A` (vector) se aranjează valorile într-o alta `B` în ordine crescătoare. Pentru aceasta se procedează astfel:

- `B[1]=A[1];`
- fiecare din valorile reținute de componentele de la `2` la `n` (în ordinea indicilor) se inserează între valorile ordonate deja ale vectorului `B`.

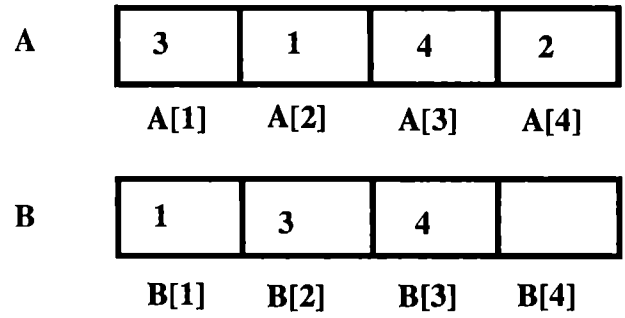
Pentru realizarea inserării unei valori, se parcurge vectorul `B` de la dreapta la stânga până când se găsește o componentă care reține o valoare mai mare (mică) decât cea care se inserează, sau până când a fost parcurs întreg vectorul. Începând cu valoarea reținută de componenta care va fi ocupată de valoarea inserată, toate valorile reținute se deplasează către dreapta cu o poziție. Fie situația inițială (în care am pus `B[1]=A[1]`):



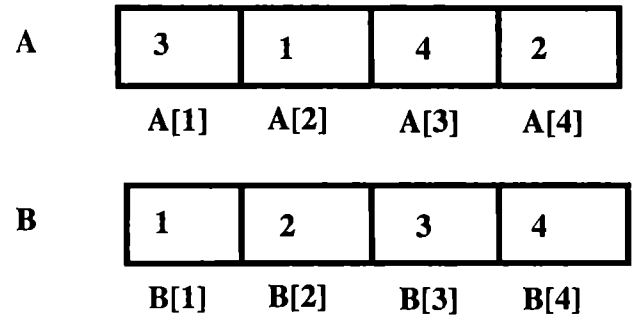
Valoarea `1` trebuie inserată înaintea valorii `3`. Pentru aceasta, `3` se deplasează la dreapta cu o poziție și pe poziția sa se inserează valoarea `1`.



Valoarea **4** trebuie inserată după **3**. Pentru aceasta nu este necesar să efectuăm nici o deplasare.



Valoarea **2** se inserează pe poziția ocupată de valoarea **3**. Pentru aceasta, valorile **3** și **4** se deplasează la dreapta cu câte o poziție.



Acum vectorul **B** reține valorile sortate. Putem să le afișăm.

```
#include <iostream.h>
int a[9],b[9],n,i,j,k,gasit;
main()
{
 cout<<"n=";cin>>n;
 for(i=0;i<n;i++)
 { cout<<"a["<<i+1<<"]=";cin>>a[i];
 };
 b[0]=a[0];
 for (i=1;i<n;i++)
  { j=i-1;
    while (j>=0 && a[i]<b[j])j--;
    for(k=i-1;k>=j+1;k--)b[k+1]=b[k];
    b[j+1]=a[i];
  }
 for(i=0;i<n;i++) cout<<b[i]<<endl;
}
```

Observații. Sortarea prin inserție poate fi făcută prin utilizarea unui singur vector. De asemenea, se face un efort de calcul pentru deplasarea la dreapta. Facem precizarea că structura de tip *vector* nu este cea indicată

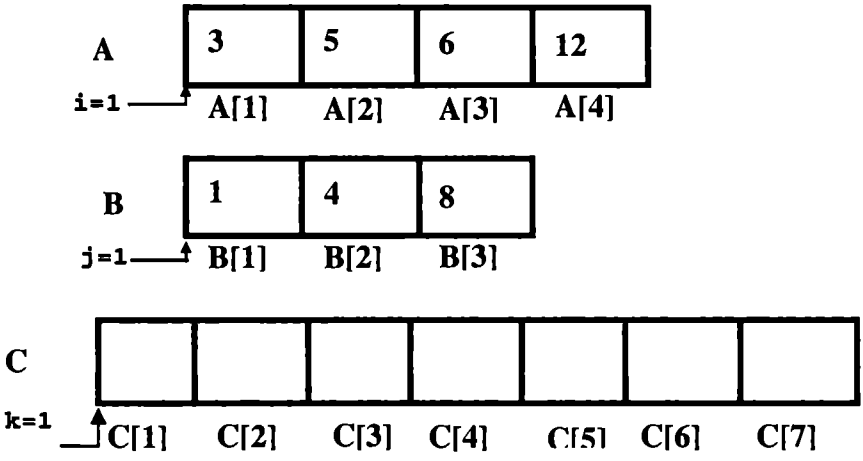
pentru acest tip de sortare. Există o altă structură care ne ajută în acest sens. Despre ea vom învăta mai târziu (clasa a XI-a).

5.3.5. Interclasare

Se citesc m numere întregi sortate crescător (descrescător). De asemenea, se citesc alte n numere întregi sortate crescător (descrescător). Se cere să se afișeze cele m+n valori în ordine crescătoare (descrescătoare).

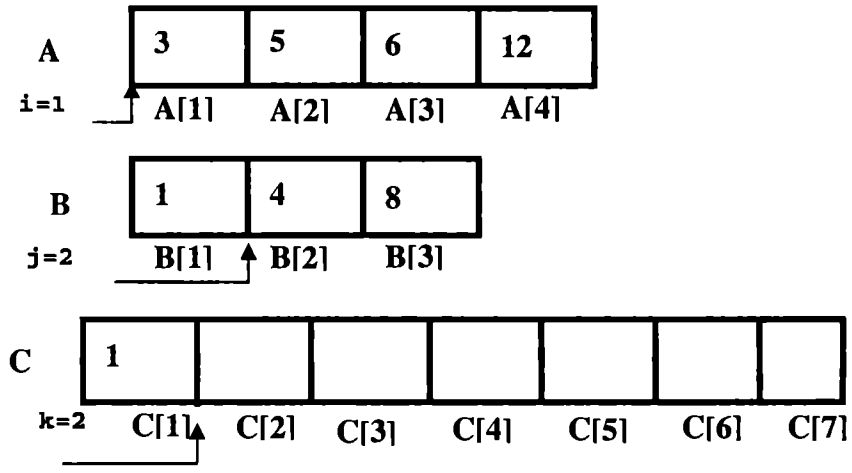
Pentru rezolvarea problemei, există un algoritm clasic, cel de *interclasare,* foarte performant. Să presupunem că cele două siruri de numere sortate se memorează în două variabile de tip vector `A` și `B`. Numerele sortate vor fi memorate într-o altă variabilă de tip vector (numită `C`). *La fiecare pas se alege un număr care va fi memorat în C (cel care urmează a fi adăugat în ordinea cerută).*

Modul de alegere rezultă din exemplul următor. Să presupunem că `m=4`, `n=3` și că au fost citite valorile:

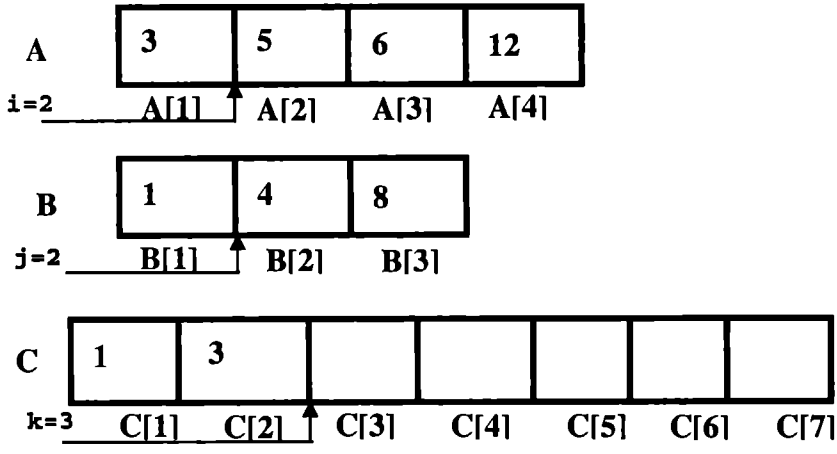


Variabila `i` reține indicele componentei din `A` care va fi comparată la pasul următor. Variabila `j` reține indicele componentei din `B` care va fi comparată la pasul următor. Variabila `k` reține indicele componentei din `C` care va memora următorul număr.

Inițial, se compară numărul reținut de `A[1]` cu cel reținut de `B[1]`. Este mai mic cel reținut de `B[1]`. Acesta este trecut în `C`, `j` va reține `2`, `i` rămâne nemodificat, iar `k` va reține `2`.



Se compară numărul reținut de `A[1]` cu cel reținut de `B[2]`. Este mai mic cel reținut de `A[1]`. Acesta este trecut în `C`, `j` va rămâne nemodificat, `i` reține `2`, iar `k` reține `3`.



Se compară `A[2]` cu `B[2]`. Este mai mic `B[2]`. `C[3]` reține `4`, `j` devine `3`, `k` va fi `4`. Se compară `A[2]` cu `B[3]`. Este mai mic `A[2]`. `C[4]` reține `5`, `k` devine `5`, iar `i` devine `3`. Algoritmul continuă în acest mod până când numerele reținute de una din variabilele `A` sau `B` sunt copiate în `C`. În continuare, se copiază în `C` numerele neanalizate. Urmăriți programul:

```
#include <iostream.h>
main()
{
 int a[100],b[100],c[200],m,n,i,j,k;
 cout<<"m=";cin>>m;

 for(i=0;i<m;i++)
 {  cout<<"a["<<i+1<<"]=";cin>>a[i] };
 cout<<"n=";cin>>n;
 for(i=0;i<n;i++)
 {
  cout<<"b["<<i+1<<"]=";cin>>b[i];
 };
 i=j=k=0;
 while(i<m && j<n)
  if (a[i]<b[j])c[k++]=a[i++];
  else c[k++]=b[j++];
 if (i<m)
  for(j=i;j<m;j++)c[k++]=a[j];
 else
  for(i=j;i<n;i++)c[k++]=b[i];
for(i=0;i<k;i++) cout<<c[i]<<endl;
}
```

5.3.6. Căutare binară

Se citesc n numere întregi sortate crescător. De asemenea se citește un număr întreg nr. Să se decidă dacă nr se găsește în șirul celor n numere citite.

Și această problemă poate fi rezolvată într-un mod elementar. Nu avem decât să comparăm, pe rând, **nr** cu toate numerele citite. Așa este, dar există un algoritm mai rapid. Acesta este algoritmul de *căutare binară* și ține cont de faptul că numerele citite sunt sortate crescător. Ideea de la care se pornește este simplă: *căutarea se efectuează între numerele reținute de componente de indice reținute de două variabile* ***li*** *și* ***ls*** *(inițial* ***li****=1 și* ***ls****=n).*

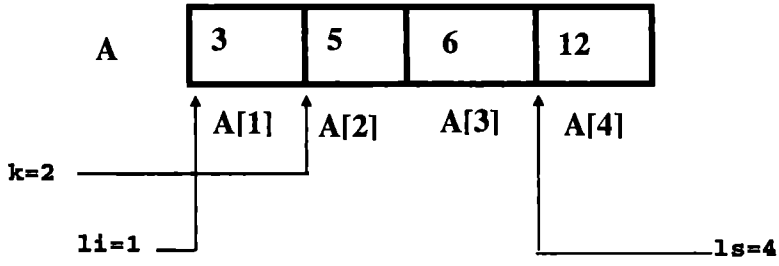
Fiind date **li** și **ls** procedăm astfel:

- se calculează indicele componentei din mijloc, în cazul în care **n** este impar, sau a uneia din cele două plasate în mijloc, în cazul în care **n** este par (**k=(li +ls)** div 2);
- apar trei posibilități:
  - valoarea reținută de componenta de indice calculat este egală cu nr (caz în care căutarea se termină cu succes);
  - valoarea reținută de componenta de indice calculat este mai mică decât nr (caz în care numărul va fi căutat între componentele de indice **li=k+1** și **ls**);

- valoarea reținută de componenta de indice calculat este mai mare decât **nr** (caz în care numărul va fi căutat între componentele de indice **li** și **ls=k-1**).

Căutarea se termină când numărul a fost identificat sau când **li>ls** (căutare fără succes).

Exemplu. Se citesc numerele de mai jos și **nr=12**. Inițial, **li=1**, **ls=4**. Avem **k=(1+4) div 2=2**.



A[2]=5<12=nr. Deci **li=k+1=3**; **ls=4**, **k=(li+ls) div 2=3**. Căutarea se face între componentele de indice **3**, **4**, iar comparația se face între **nr** și **A[3]**.

```
#include <iostream.h>
main()
{
 int a[100],n,i,li,ls,k,nr,gasit;
 cout<<"n=";cin>>n;
 for(i=0;i<n;i++)
 {
  cout<<"a["<<i+1<<"]=";cin>>a[i];
 };
 cout<<"nr=";cin>>nr;
 li=0;ls=n-1;gasit=0;
 do
  {
   k=(li+ls)/2;
   if(a[k]==nr)
    {
     cout<<"gasit pe pozitia "<<k+1;
     gasit=1;
    }
   else
    if (a[k]<nr)li=k+1;
    else ls=k-1;
  } while (li<=ls && !gasit);
  if(li>ls)cout<<"negasit";
}
```

5.4. Aplicații cu matrice

1. Interschimbarea liniilor. *Se citește un tablou cu* `m` *linii și* `n` *coloane. Se citesc de asemenea și două numere naturale, distincte,* `x` *și* `y`*, cuprinse între* `1` *și* `m`*. Se cere să se interschimbe linia* `x` *cu linia* `y`*. La început vom afișa tabloul inițial, apoi pe cel obținut prin interschimbarea liniilor* `x` *și* `y`*.*

Exemplu. **`m=3, n=3, x=2, y=3,`** iar tabloul este:

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$$

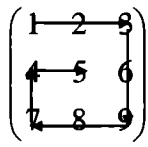
În urma interschimbării liniilor 2 și 3 se obține:

$$\begin{pmatrix} 1 & 2 & 3 \\ 7 & 8 & 9 \\ 4 & 5 & 6 \end{pmatrix}$$

După cum am învățat, pentru a interschimba conținutul a două variabile se utilizează o a treia, de manevră. De această dată se interschimbă două linii. Am fi tentați ca manevra să fie un vector cu **`n`** componente. Cu puțină atenție ne dăm seama că putem folosi ca manevră o singură variabilă de același tip cu cel al componentelor de bază ale tabloului. În rest, analizați programul:

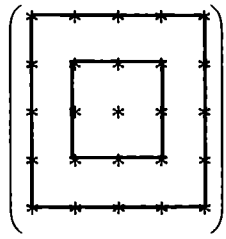
```
#include <iostream.h>
main()
{ int mat[10][10],m,n,i,j,x,y,man;
  cout<<"m=";cin>>m;
  cout<<"n=";cin>>n;
  for(i=0;i<m;i++)
   for(j=0;j<n;j++)
    { cout<<"mat["<<i+1<<','<<j+1<<"]=";cin>>mat[i][j];
    };
  cout<<"x=";cin>>x;
  cout<<"y=";cin>>y;
  cout<<endl;
  for (i=0;i<m;i++)
  { for (j=0;j<n;j++)cout<<mat[i][j]<<' ';
    cout<<endl;
  }
  for(j=0;j<n;j++)
  { man=mat[x-1][j];
    mat[x-1][j]=mat[y-1][j];
    mat[y-1][j]=man;
  }
  cout<<endl;
  for (i=0;i<m;i++)
  { for (j=0;j<n;j++)cout<<mat[i][j]<<' ';
    cout<<endl;
  }}
```

2. Spirala. *Se citește un tablou cu* **n** *linii și* **n** *coloane. Se cere să se afișeze elementele tabloului în ordinea rezultată prin parcurgerea acestuia în spirală, începând cu primul element din linia 1 în sensul acelor de ceas.* Exemplu. Fie tabloul:



Elementele vor fi afișate în ordinea: **1 2 3 6 9 8 7 4 5**.

Pentru a putea rezolva problema, privim tabloul ca pe un ansamblu alcătuit din mai multe dreptunghiuri "concentrice".



Tabloul de mai sus este un ansamblu format din trei dreptunghiuri "concentrice" - ultimul are un singur element și nu a putut fi reprezentat. După ce stabilim numărul de pătrate (cum?) afișăm elementele aflate pe fiecare latură a fiecărui pătrat în ordinea cerută, având grijă ca elementele aflate în colțuri să nu fie afișate de două ori.

```
#include <iostream.h>
main()
{
 int mat[10][10],n,i,j,k;
 cout<<"n=";cin>>n;
 for(i=1;i<=n;i++)
  for (j=1;j<=n;j++)
    { cout<<"mat["<<i<<','<<j<<"]=";cin>>mat[i][j]; }
 for(k=1;k<=n/2+1;k++)
 {
  for(i=k;i<=n-k+1;i++)cout<<mat[k][i]<<endl;
  for(i=k+1;i<=n-k+1;i++)cout<<mat[i][n-k+1]<<endl;
  for(i=n-k;i>=k;i--)cout<<mat[n-k+1][i]<<endl;
  for(i=n-k;i>=k+1;i--)cout<<mat[i][k]<<endl;
 }
}
```

Observați faptul că, de această dată am lucrat cu indicii cuprinși între **1** și **n**. Aceasta înseamnă că n-am utilizat eficient memoria, pentru că spațiul necesar liniei **0** și coloanei **0** a rămas liber. În schimb, algoritmul a fost elaborat mai ușor. Dacă programul nu are nevoie de multă memorie se poate lucra și așa. Refaceți acest program pentru a lucra cu indici între **0** și **n-1**.

5.5 Sortarea fără comparații

Este o metodă de sortare care permite sortarea a **n** numere naturale fără a face nici măcar o comparație între ele.

Vom prezenta algoritmul pe un exemplu, în care se sortează crescător **6** numere naturale: **6, 36, 41, 25, 40, 30**.

Inițial, șirul celor **n** numere se împarte în **10** clase: prima clasă conține numerele care se termină cu **0**, a 2-a clasă cele care se termină cu **1**,... a **10** clasă cele care se termină cu **9**.

Pentru memorarea numerelor care aparțin fiecărei clase vom utiliza o matrice **Mat** cu **10** coloane, în care prima linie are indicele **0**. Elementele după linia **0** reține numărul de elemente din șir care se găsesc pe coloana respectivă.

0	1	2	3	4	5	6	7	8	9
2	1	0	0	0	1	2	0	0	0
40	41				25	6			
30						36			

Obținem din nou șirul de numere, așezându-le în ordinea coloanelor și în ordinea în care le-am pus în fiecare coloană: **40, 30, 41, 25, 6, 36.** Împărțim, din nou, șirul în 10 clase după a 2-a cifră a numerelor.

0	1	2	3	4	5	6	7	8	9
1	0	1	2	2	0	0	0	0	0
6		25	30	40					
			36	41					

Obținem din nou șirul de numere, așezându-le în ordinea coloanelor și în ordinea în care le-am pus în fiecare coloană: **6, 25, 30, 36, 40, 41**. De această dată șirul este sortat.

✓ Dacă **k** este numărul maxim de cifre a numerelor din șir, operația de împărțire a numerelor în **10** clase se va relua de **k** ori.

✓ Algoritmul este "mare consumator" de memorie!

```
#include <iostream.h>
int A[100],Mat[100][100],n,NrCif,i,j,k,Cif,Zece;

main()
{ cout<<"n=";cin>>n;
  for (i=1;i<=n;i++)
   {
    cout<<"A["<<i<<"]=";cin>>A[i];
   }
   Zece=1;
   for(NrCif=1;NrCif<=4;NrCif++)
   {
    if (NrCif>1) Zece*=10;
    for (i=1;i<=n;i++)
    {
      Cif= (A[i] / Zece) % 10;
      Mat[0][Cif]++;
      Mat[ Mat[0][Cif] ][Cif]=A[i];
    }

  //Extrag din matrice in vector
   k=0;
   for (i=0;i<=9;i++)
   if (Mat[0][i])
    for (j=1;j<=Mat[0][i];j++)
    {
      k++;
      A[k]=Mat[j][i];
     }
   for (i=0;i<=9;i++) Mat[0][i]=0;
  }
 for (i=1;i<=n;i++) cout<<A[i]<<endl;
}
```

Probleme propuse

1. Să se citească și să se afișeze un vector cu componente de tip **int**.

2. Să se citească și să se afișeze o matrice cu componente de tip **float**.

3. Să se citească și afișeze un vector cu **n** caractere (exclusiv litere). Afișați și codurile caracterelor.

4. Sortați vectorul de la problema precedentă după codurile caracterelor. Afișați caracterele sortate. Ce observați?

5. Să se citească, pe litere, un cuvânt. Programul va verifica dacă în cuvântul citit toate literele sunt distincte. Exemple: **martie** - are litere distincte; **parcare** - nu are litere distincte.

6. Se citește un vector cu **n** componente, numere naturale. Să se afișeze cel mai mare număr rațional subunitar în care numărătorul și

numitorul fac parte din mulțimea valorilor citite. Exemplu: dacă am citit valorile **1 2 3** se afișează **2/3**.

7. Același enunț ca la problema anterioară, numai că se cere să se afișeze cel mai mic număr rațional supraunitar.

8. Se citește un vector cu **n** componente numere întregi. Care este cea mai mare sumă care se poate forma cu ele? Exemplu: dacă **n=4**, iar numerele citite sunt **-1 3 2 -7**, se afișează **5**.

9. Se citește un vector cu n componente numere întregi distincte și un număr natural **S**. Să se găsească (dacă există) o submulțime a numerelor citite pentru care suma elementelor este **S**.

10. Se citește un vector cu n componente numere naturale. Se cere să se obțină toate permutările circulare la dreapta. Exemplu: dacă **n=4**, și vectorul este **1 2 3 4**, permutările circulare sunt:

```
1 2 3 4
4 1 2 3
3 4 1 2
2 3 4 1.
```

Observați faptul că, de fiecare dată, ultimul element trece pe prima poziție, iar restul elementelor sunt deplasate la dreapta cu o poziție.

11. Se citesc un număr natural **n** și doi vectori **a** și **b** cu **n** componente numere întregi. Se cere să se calculeze suma:

$$S = \frac{a[1]}{b[1]} + \frac{a[2]}{b[2]} + \ldots \frac{a[n]}{b[n]}.$$

12. Se dau **n** numere raționale, reținute în doi vectori **a** și **b**. Astfel, numărătorul primului număr este **a[1]**, numitorul său este **b[1]**, etc. Programul va decide dacă numerele date sunt distincte. În cazul în care răspunsul este afirmativ, programul va afișa **DA**, altfel va afișa **NU** și o pereche de numere egale.

13. Se dau n numere raționale *distincte*, reținute în doi vectori **a** și **b** și **m** numere raționale distincte reținute în vectorii **c** și **d**. Se consideră că cele **n** numere alcătuiesc o mulțime **X** și cele m numere alcătuiesc o mulțime **Y**. Se cere să se afișeze:

- **X∪Y;**
- **X∩Y;**
- **X-Y.**

Observație: pentru fiecare cerință se va face un program.

14. Se dau **n** numere raționale, reținute în doi vectori **a** și **b**. Se cere să se sorteze descrescător, utilizând toate metodele de sortare cunoscute. Observație: pentru fiecare cerință se va face un program.

15. Se dau n numere raționale *distincte*, reținute în doi vectori **a** și **b** și **m** numere raționale distincte reținute în vectorii **c** și **d**. Atât cele **n** numere reținute în vectorii **a** și **b**, cât și cele **m** numere reținute în vectorii **c** și **d** sunt sortate crescător. Se cere să se *interclaseze* cele două șiruri de numere.

16. Se citește un vector cu **n** componente numere naturale. Să se aranjeze numerele în vector astfel încât cele pare să ocupe primele poziții. Exemplu. Dacă **n=4** și se citește 1 2 7 8, o soluție este 2 8 1 7.

17. Același enunț ca la problema precedentă, numai că se citesc litere și se cere ca vocalele să fie plasate pe primele poziții.

18. Se citește un număr natural **n** (citirea se va face într-o variabilă de tip **int**). Să se scrie acest număr într-un vector. Exemplu: Dacă citim **1231**, vom avea **V[1]=1, V[2]=2, V[3]=3, V[4]=1**.

19. Se citește un vector cu **n** componente numere naturale. Să se afișeze cel mai mare divizor al celor **n** numere.

20. Se citește un vector cu componente numere naturale. Să se afișeze numărul cifrelor 0 cu care se termină numărul format din produsul celor **n** componente.

Indicație. Nu putem efectua produsul celor **n** numere pentru ca, apoi, să vedem cu câte cifre **0** se termină, pentru că într-un astfel de caz numărul obținut poate fi foarte mare (nu poate fi reținut într-o variabilă de tip **int** sau **long**). Soluția este următoarea:

- se determină k_1, exponentul maxim al cifrei **2** astfel încât 2^{k1} divide pe **V[1]**;
- ...;
- se determină k_n, exponentul maxim al cifrei **2**, astfel încât 2^{kn} divide **V[n]**;
- se calculează $a=k_1+k_2+\ldots+k_n$ (am determinat puterea maximă a lui 2 astfel încât 2^a divide produsul considerat);
- se determină p_1, exponentul maxim al cifrei **5**, astfel încât 5^{p1} divide pe **V[1]**;
- . . .
- se determină p_n, exponentul maxim al cifrei **5**, astfel încât 5^{pn} divide **V[n]**;
- se calculează $b=p_1+p_2+\ldots+p_n$ puterea maximă a lui **5** astfel încât 5^b divide produsul considerat.

Numărul de cifre 0 cu care se termină produsul va fi minimul între **a** și **b** (de ce?).

21. Să se calculeze 2^{100}.

Indicație. Dacă am calcula produsul pe căi clasice, numărul obținut este atât de mare încât nu se poate reprezenta în memorie nici chiar prin utilizarea tipului **longint**. Pentru aceasta, suntem nevoiți să simulăm înmulțirea cu 2, reținând cifrele numărului într-un vector.

Astfel, se pornește de la un vector ale cărui componente au valorile:

0 0 0 0 0...0 0 0 1 (numărul **1**);

- se înmulțește fiecare componentă a vectorului cu 2:

```
0 0 0 0 0...0 0 0 2;
0 0 0 0 0...0 0 0 4;
0 0 0 0 0...0 0 0 8;
0 0 0 0 0...0 0 0 16;
```

- după fiecare înmulțire cu **2** se parcurge vectorul în sens invers și pentru fiecare componentă (**n**) cu un conținut mai mare decât 9 se scade 10 și se adună 1 la componenta **n-1**:

```
0 0 0 0 0...0 0 1 6
```

- se va obține în continuare:

```
0 0 0 0 0...0 0 2 12;
0 0 0 0 0...0 0 3 2;
0 0 0 0 0...0 0 6 4
```

Estimați numărul de cifre pe care îl va avea numărul 2^{100} astfel încât să reținem tipul vector cu atâtea componente câte sunt necesare.

22. Un număr natural se reține într-un vector astfel încât fiecare componentă a vectorului conține o cifră a numărului. Exemplu: dacă vectorul **V** are **5** componente, iar numărul este **128**, **V[1]=0**, **V[2]=0**, **V[3]=1**, **V[4]=2**, **V[5]=8**. Să se înmulțească numărul cu alt număr natural format dintr-o singură cifră.

23. Aceeași problemă ca și precedenta, însă înmulțitorul are mai multe cifre și este reprezentat într-un vector.

24. Se citesc într-un vector **n** numere naturale: $a_1, a_2, \ldots, a_n$. Să se calculeze:

$S_1=a_1+a_2+\ldots+a_n;$

$S_2=a_1a_2+\ldots+a_1a_n+a_2a_3+\ldots+a_2a_n+\ldots+a_{n1}a_n;$

...

$S_n=a_1a_2\ldots a_n.$

Indicație. Sumele considerate sunt coeficienții polinomului:

$P=(x+a_1)(x+a_2)\ldots(x+a_n)$. Astfel:

$(x+a_1)(x+a_2)=x^2+(a_1+a_2)x+a_1a_2$

$(x+a_1)(x+a_2)(x+a_3)=x^3+(a_1+a_2+a_3)x^2+(a_1a_2+a_1a_3+a_2a_3)x+a_1a_2a_3$

$(x+a_1)(x+a_2)(x+a_3)\ldots(x+a_n)=x^n+(a_1+\ldots+a_n)x^{n-1}+$

$(a_1a_2+\ldots+a_{n-1}a_n)x^{n-2}+\ldots+a_1a_2\ldots a_n$

25. Se consideră **F** un polinom de gradul **n**, unde cei **n+1** coeficienți sunt reținuți într-un vector. Se citește o valoare reală **a**. Să se afișeze **F(a)**.

26. Se dau două polinoame ai căror coeficienți sunt reținuți în doi vectori. Să se calculeze coeficienții produsului celor două polinoame.

27. Se consideră un vector care conține **n** numere naturale. Să se afișeze o submulțime a acestora a cărei sumă se divide cu **n**.

Indicație. Există întotdeauna o secvență de numere consecutive a căror sumă se divide la **n**. Să demonstrăm acest fapt. Calculăm sumele:

$S_1 = a_1;$
$S_2 = a_1+a_2;$
. . .
$S_n = a_1+a_2+\ldots+a_n.$

Dacă suma S_k se divide cu **n** problema este rezolvată (se tipăresc $a_1, a_2, \ldots, a_k$). Să presupunem că nici o sumă nu se divide cu **n**. Știm de la matematică faptul că restul împărțirii la **n** poate fi **0, 1,..., n-1**. Cazul restului **0** a fost tratat anterior. Rezultă că nu putem avea decât un rest cuprins între **1** și **n-1**. Întrucât avem **n** sume și **n-1** posibilități de rest, rezultă că există două sume care dau același rest la împărțirea la **n**. Fie acestea S_i și S_j **(i<j)**.

$S_i = a_1+a_2+\ldots+a_i;$
$S_j = a_1+a_2+\ldots+a_i+\ldots+a_j;$
$S_j - S_i = a_{i+1}+\ldots+a_j.$

Se tipăresc $a_{i+1}, a_{i+2}, \ldots, a_j$ (diferența a două numere care dau același rest prin împărțirea cu **n** se divide la **n**).

28. Se citește **n**, număr natural. Se cere să găsim o alegere a semnelor ±, dacă este posibil, astfel să avem relația $1\pm2\pm3\pm4\pm\ldots\pm n=0$. În caz afirmativ se va afișa expresia corectă, altfel programul va scrie **NU**.

Liviu Panaitopol.

Indicație. Avem **1+2-3=0** și **1-2-3+4=0**. Din relația 2 se deduce cu ușurință: **(n+1)-(n+2)-(n+3)+(n+4)=0**. Deci dacă **n** este de forma **4k** sau **4k+3** alegerea semnelor este imediată.

Fie **n=4k+1**. Atunci suma este:

(1± 2± 3 ± 4)±(5 ± 6 ± 7 ± 8)±.... ±.(4k-3 ± 4k-2 ± 4k-1 ± 4k) +4k+1.

Fiecare termen cuprins între paranteze este par. Dacă la prima valoare se adună sau se scade a doua valoare se obține o valoare impară, apoi dacă la aceasta se adună sau se scade a treia valoare se obține o valoare pară și dacă la aceasta se adună sau se scade a patra valoare se obține o valoare pară. Suma termenilor din paranteze este o valoare pară și ultima valoare este impară. Prin urmare, oricare ar fi alegerea semnelor rezultatul este impar. Ori, un impar nu poate fi egal cu 0. Programul va afișa **NU**.

Dacă **n** este de forma **4k+2** atunci putem grupa astfel termenii:

(1± 2) ± (3 ± 4) ±(5 ± 6) ± .... ±(4k+1 ± 4k+2).

Indiferent de alegerea semnelor, fiecare termen încadrat de paranteze ia o valoare impară. Avem **2k+1** perechi. Pentru orice alegere a semnelor suma primelor **k** va fi un număr par. Cum ultima pereche este impară rezultatul este impar, deci nu poate fi **0**.

29. Problema concursului. La un concurs se întâlnesc **n** jucători. Fiecare jucător trebuie să joace o singură dată cu fiecare din ceilalți **n-1** jucători. Se cere să se elaboreze un program al întâlnirilor.

Problemă rezolvată de la Orest Bolohan (Suceava)

Indicație. Numărul de jucători trebuie să fie par. Dacă **n** nu este par, se adaugă un jucător fictiv (cel care joacă cu el stă degeaba). Avem **n-1** etape (în fiecare etapă jucătorul **i** joacă cu unul din ceilalți **n-1** jucători). Într-o etapă se organizează **n/2** jocuri. Deci, se organizează **n(n-1)/2** jocuri.

Într-un vector sunt trecuți toți cei **n** jucători. Presupunem **n=4**. Vectorul este:

```
1 2 3 4
```

Vectorul se împarte în două și se așează de așa natură încât primul jucător este lângă ultimul, al doilea lângă penultimul etc.:

```
1 2
4 3
```

În etapa 1 se organizează partidele: **1-4; 2-3.** Șirul se permută circular de la dreapta la stânga cu excepția ultimului element.

```
1 3 4 2
```

Procedăm apoi ca înainte:

```
1 3
2 4
```

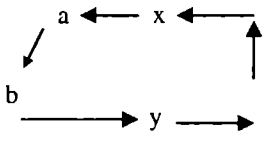
În etapa 2 se organizează partidele: 1-2; 3-4. Șirul se permută circular de la dreapta la stânga cu excepția ultimului element.

```
1 4 2 3
```

Procedăm ca înainte:

```
1 4
3 2
```

În etapa 3 se organizează partidele: `1-3; 4-2.`



În acest fel se organizează `n(n-1)/2` partide. Trebuie demonstrat că sunt distincte. Adică dacă `x` se întâlnește cu `y` nu vom avea și că `y` se întâlnește cu `x`. Dacă `x` se întâlnește cu `y`, `x` este deasupra (în vectorul împărțit).

În stânga perechii `(x,y)` sunt jucători "permutabili" `(a, b,...)` în număr impar (pentru că `1` nu este permutabil). În dreapta perechii `(x,y)` sunt jucători "permutabili" în număr par. Dacă `y` ajunge deasupra lui `x`, ținând cont de logica algoritmului, înseamnă că toți jucătorii, care inițial erau în stânga perechii `(x,y)`, sunt acum în dreapta. Dar numărul lor era par și a ajuns impar. Absurd!

30. Se citește un tablou cu `n` linii și `n` coloane, numere întregi. Un astfel de tablou, în care numărul liniilor coincide cu numărul coloanelor, se numește *tablou pătratic*.

a) Pentru un tablou pătratic A, numim diagonală principală, elementele aflate pe "linia" care unește `A[1,1]`, `A[n,n]`. Exemplu: pentru tabloul alăturat, elementele sunt: `1, 5` și `9`.

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$$

Se cere:

a1) suma elementelor aflate pe diagonala principală;
a2) suma elementelor aflate deasupra diagonalei principale;
a3) suma elementelor aflate sub diagonala principală.

b) Pentru un tablou pătratic A, numim diagonală secundară, elementele aflate pe "linia" care unește `A[n][1]`, `A[1][n]`. Exemplu: pentru tabloul alăturat, elementele sunt: `7, 5` și `3`.

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$$

Se cere:

b1) suma elementelor aflate pe diagonala secundară;
b2) suma elementelor aflate deasupra diagonalei secundare;
b3) suma elementelor aflate sub diagonala secundară.

Cerințe suplimentare.

1. Pentru fiecare cerință se va face un program separat.
2. În nici un program nu se va folosi instrucțiunea `if`.

31. Interschimbați coloanele unei matrice cu `m` linii și `n` coloane astfel încât în linia `k` elementele să fie în ordine crescătoare.

32. Interschimbați linii și coloane ale unei matrice cu **n** linii și **n** coloane astfel încât elementele de pe diagonala principală să fie sortate crescător.

33. Un teren este dat sub forma unui tablou **A** cu **n** linii și **m** coloane. Elementul **A[i][j]** reține altitudinea pătrățelului de coordonate **i** și **j**. Să se afișeze coordonatele "pătrățelelor vârf" (un pătrățel este vârf dacă toți vecinii săi au o altitudine strict mai mică).

34. Determinați elementele șa ale unei matrice cu **n** linii și **m** coloane (elemente minime pe linie și maxime pe coloană sau maxime pe linie și minime pe coloană).

Indicație. Atenție! Maximele sau minimele pe linii sau coloane pot să nu fie unice. Dacă vom considera numai o valoare dintre acestea, s-ar putea să pierdem soluții.

Testele de la **35** la **38** se referă la algoritmul următor:

```
Citește n
Pentru i ← 1, n execută
|  Citește V[i] (V[i] este număr natural)
|_■
s ← 0
Pentru i ← 1, n execută
|  Cât timp V[i]≥10 execută
|  |  V[i]← V[i] div 10;
|  |_■
|  s ← s + V[i]
|_■
Scrie S
```

35. Ce se afișează dacă **n=4** și **V=(21,1,7,61)**?

a) 16 **b) 7** **c) 17** **d) 10.**

36. Ce se afișează dacă **n=4** și **V=(01,1,07,1)**?

a) 01 **b) 10** **c) 011** **d) 4.**

37. Pentru care dintre vectorii de mai jos, în urma prelucrării se afișează **9**?

a) n=3 V=(9, 9, 9);
b) n=4 V=(4, 5, 6, 7);
c) n=4 V=(11, 12,13, 19);
d) n=1 V=(933).

38. Dacă dorim să obținem un program C++ și V trebuie să rețină cel mult 19 numere naturale între 0 și 300, care dintre declarațiile de mai jos este corectă?

a) int V[19];
b) char V[19];
c) double V[19];
d) int V[20].

Testele de la 39 la 42 se referă la algoritmul următor:

```
Citeşte n
Pentru i ← 1, n execută
|  Citeşte V[i] (V[i] este număr natural)
|_■
i ← 2
Valoare ← V[1]
Scrie Valoare
Repetă
|  Cât timp (Valoare≤V[i]) SI (i≤n) execută
|  |  i ← i + 1
|  |_■
|  Dacă i≤n atunci
|  |     Valoare ← V[i]
|  |     i ← i+1
|  |     Scrie Valoare
|  |_■
|  Cât timp (Valoare≥V[i]) SI (i≤n) execută
|  |  i ← i + 1
|  |_■
|  Dacă i≤n atunci
|  |     Valoare ← V[i]
|  |     i ← i+1
|  |     Scrie Valoare
|  |_■
|_■ până când i>n
```

39. Ce se afișează dacă se citește n=4 și 12 13 11 15?

a) 12 13 11 b) 12 11 15 c) 12 13 11 d) 12 13 11 15.

40. Pentru care dintre vectorii următori algoritmul afișează doar valoarea maximă din vector?

a) n=4 V=(5, 4, 2, 3);
b) n=4 V=(5, 7, 2, 3);
c) nu există un astfel de vector;
d) nici una din variantele de mai sus nu este adevărată.

41. Pentru care dintre vectorii următori algoritmul afișează doar valoarea minimă din vector?

a) n=4 V=(3, 7, 4, 5);
b) n=4 V=(5, 7, 2, 3);
c) nu există un astfel de vector;
d) nici una din variantele de mai sus nu este adevărată.

42. Pentru care dintre vectorii următori algoritmul afișează toate elementele?

a) n=4 V=(3, 7, 4, 5);
b) n=4 V=(5, 7, 2, 3);
c) nu există un astfel de vector;
d) n=4 V=(7, 6, 7, 6).

43. Care este șirul celor trei numere afișate de calculator, dacă primul dintre ele este 97?

```
#include <iostream.h>
int i; char V[10];
main()
{  V[1]='a'; V[2]='d'; V[3]='c';
   for (i=1;i<=3;i++) cout<<(int)V[i]<<" ";}
```

44. Ce afișează programul de mai jos?

```
#include <iostream.h>
int i,V[10];
main()
{  V[1]=1; V[2]=3; V[3]=3;
   for (i=1;i<=4;i++) cout<<V[i];
}
```

a) 1330 b) 133 c) 133 d) Mesaj de eroare.

45. Ce se afișează dacă n=4 și V=(2, 4, 1, 5) ?

```
#include <iostream.h>
int n, i, V[10],V1[10];
main()
{  cout<<"n=";cin>>n;
   for (i=1;i<=n;i++) cin>>V[i];
   for (i=1;i<n;i++) V1[i]=V[i]<V[i+1];
   for (i=1;i<=n;i++)
    if (V1[i]) cout<<V[i]<<" ";
}
```

a) 2 4 1 5;
b) 2 1;
c) 4 5;
d) 2 1 5.

46. Se citește un vector **V**, cu elemente numere naturale. Se cere să se înlocuiască fiecare element al său cu media aritmetică dintre elementul anterior și cel care îi urmează. Se vor excepta de la această prelucrare primul și ultimul element. Care dintre programele de mai jos este corect?

a)
```
#include <iostream.h>
int n, i,V[10],V1[10];
main()
{ cout<<"n=";cin>>n;
 for (i=1;i<=n;i++) cin>>V[i];
 for(i=2;i<n;i++)
  V1[i]=(V[i-1]+V[i+1])/2;
 for(i=2;i<n;i++) V[i]=V1[i];
 for(i=1;i<=n;i++)
 cout<<V[i]<<" ";
}
```

b)
```
#include <iostream.h>
int n, i; float V[10];
main()
{ cout<<"n=";cin>>n;
 for (i=1;i<=n;i++) cin>>V[i];
 for(i=2;i<n;i++)
  V[i]=(V[i-1]+V[i+1])/2;
 for(i=1;i<=n;i++)
 cout<<V[i]<<" ";
}
```

c)
```
#include <iostream.h>
int n, i; float V[10],V1[10];
main()
{ cout<<"n=";cin>>n;
 for (i=1;i<=n;i++) cin>>V[i];
 for(i=2;i<n;i++)
  V1[i]=(V[i-1]+V[i+1])/2;
 for(i=2;i<n;i++) V[i]=V1[i];
 for(i=1;i<=n;i++)
 cout<<V[i]<<" ";
}
```

d)
```
include <iostream.h>
int n, i, V[10];
main()
{ cout<<"n=";cin>>n;
 for (i=1;i<=n;i++) cin>>V[i];
 for(i=2;i<n;i++)
  V[i]=(V[i-1]+V[i+1])/2;
 for(i=1;i<=n;i++)
 cout<<V[i]<<" ";
}
```

47. Care dintre secvențele de mai jos permută circular la dreapta elementele unui vector V? De exemplu, dacă **V=(1,2,3,4)**, după prelucrare trebuie ca **V=(4,1,2,3)**.

a)
```
Man=V[n];
for (i=1;i<n;i++)
              V[i+1]=V[i];
V[1]=Man;
```

b)
```
Man=V[n];
 for(i=n;i>=1;i--)
          V[i+1]=V[i];
V[1]=Man;
```

c)
```
Man=V[1];
 for(i=n;i>=1;i--)
          V[i+1]=V[i];
V[1]=Man;
```

d)
```
Man=V[n];
for (i=1;i<n;i++)
              V[i]=V[i+1];
V[1]=Man;
```

48. Fiind dată o matrice cu **n** linii și **n** coloane (pătratică) cu numere naturale și fiind date două elemente ale matricei de coordonate **(x1,y1)** și **(x2,y2)** care dintre relațiile de mai jos testează dacă elementele se găsesc pe o dreaptă paralelă cu una dintre diagonalele matricei (principală sau secundară)?

```
a)  if (x1-x2==y1-y2) cout<<"Da";
                      else cout<<"Nu";

b)  if (x1-y1==x2-y2) cout<<"Da"
                      else cout<<"Nu";

c)  if (abs(x1-y1)==abs(x2-y2)) cout<<"Da";
                                else cout<<"Nu";

d)  if (abs(x1-x2)==abs(y1-y2)) cout<<"Da";
                                else  cout<<"Nu";
```

49. Fereastra. Fiind dată o matrice cu **m** linii și **n** coloane, se cere să se afișeze toate submatricele cu **3** linii și **3** coloane ale matricei inițiale. Un astfel de procedeu este utilizat atunci când, de exemplu, o imagine este mult prea mare și ea este afișată cu ajutorul unei ferestre. Exemplu: **m=4**, **n=4**. Matricea inițială și submatricele sunt prezentate mai jos:

```
1 2 3 4
5 6 7 8          1 2 3     2 3 4     5 6 7     6 7 8
9 10 11 12       5 6 7     6 7 8     9 10 11   10 11 12
13 14 15 16      9 10 11   10 11 12   13 14 15  14 15 16
```

Care dintre secvențele de mai jos este corectă?

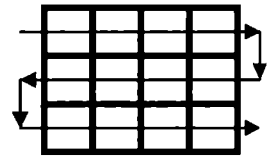
```
a)
  for (i=1;i<=m-2;i++)
   for (j=1;j<=n-2;j++)
   {cout <<"Matrice"<<endl;
    for(l=0;l<=2;l++)
    { for(c=0;c<=2;c++)
      cout<<Mat[i+l][j+c]<<' ';
      cout<<endl;
    }
   }
```

```
b)
for (i=1;i<=n-2;i++)
   for (j=1;j<=m-2;j++)
   {cout <<"Matrice"<<endl;
    for(l=0;l<=2;l++)
    { for(c=0;c<=2;c++)
      cout<<Mat[i+l][j+c]<<' ';
      cout<<endl;
    }
   }
```

```
c)
for (i=1;i<=m-2;i++)
   for (j=1;j<=n-2;j++)
   {cout <<"Matrice"<<endl;
    for(l=1;l<=3;l++)
    { for(c=1;c<=3;c++)
      cout<<Mat[i+l][j+c]<<' ';
      cout<<endl;
    }
   }
```

```
d)
for (i=1;i<=n-2;i++)
   for (j=1;j<=m-2;j++)
   {cout <<"Matrice"<<endl;
    for(l=1;l<=3;l++)
    { for(c=1;c<=3;c++)
      cout<<Mat[i+l][j+c]<<' ';
      cout<<endl;
    }
   }
```

50. Fiind dată o matrice **Mat**, cu **m** linii și **n** coloane, care dintre secvențele următoare afișează elementele matricei în sensul indicat în figura alăturată?



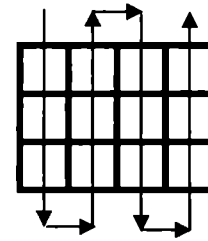
a)
```
sens=1;
for (i=1;i<=n;i++)
{ if (sens==1)
  for (j=1;j<=m;j++)
   cout<<Mat[i][j]<<' ';
  else
   for (j=n; j>=1;j--)
    cout<<Mat[i][j]<<' ';
  sens=-sens;
 }
```

b)
```
sens=1;
for (i=1;i<=m;i++)
{ if (sens=1)
  for (j=1;j<=n;j++)
   cout<<Mat[i][j]<<' ';
  else
   for (j=n; j>=1;j--)
    cout<<Mat[i][j]<<' ';
  sens=-sens;
 }
```

c)
```
sens=-1;
for (i=1;i<=m;i++)
{ if (sens==1)
  for (j=1;j<=n;j++)
   cout<<Mat[i][j]<<' ';
  else
   for (j=n; j>=1;j--)
    cout<<Mat[i][j]<<' ';
  sens=-sens;
 }
```

d)
```
sens=1;
for (i=1;i<=m;i++)
{ if (sens==1)
  for (j=1;j<=n;j++)
   cout<<Mat[i][j]<<' ';
  else
   for (j=n; j>=1;j--)
    cout<<Mat[i][j]<<' ';
  sens=-sens;
 }
```

51. Fiind dată o matrice **Mat**, cu **m** linii și **n** coloane, care dintre secvențele de mai jos afișează elementele matricei în sensul indicat în figura alăturată?



a)
```
sens=1;
for (i=1;i<=n;i++)
{ if (sens==1)
  for (j=1;j<=m;j++)
   cout<<Mat[j][i]<<' ';
  else
   for (j=m;j>=1;j--)
    cout<<Mat[j][i]<<' ';
   sens=-sens;
}
```

b)
```
sens=1;
for (i=1;i<=n;i++)
{ if (sens==1)
  for (j=1;j<=m;j++)
   cout<<Mat[i][j]<<' ';
  else
   for (j=m;j>=1;j--)
    cout<<Mat[i][j]<<' ';
   sens=-sens;
}
```

c)
```
sens=-1;
for (i=1;i<=n;i++)
{ if (sens==1)
  for (j=1;j<=m;j++)
   cout<<Mat[j][i]<<' ';
  else
   for (j=m;j>=1;j--)
    cout<<Mat[j][i]<<' ';
   sens=-sens;
}
```

d)
```
sens=1;
for (i=1;i<=m;i++)
{ if (sens==1)
  for (j=1;j<=n;j++)
   cout<<Mat[j][i]<<' ';
  else
   for (j=n;j>=1;j--)
    cout<<Mat[j][i]<<' ';
   sens=-sens;
}
```

52. Se dă un vector **v** cu **m*n** componente numere întregi. Care dintre secvențele de mai jos copiază vectorul **v** în matricea **Mat** cu **m** linii și **n** coloane. Copierea se realizează completând mai întâi elementele de pe linia **1**, apoi elementele de pe linia **2**,... la sfârșit elementele de pe linia **m**.

✓ Copierea se realizează prin utilizarea unui singur ciclu **for**!

```
a) for (k=1;k<=m*n;k++)  Mat[1+(k-1)/n][1+(k-1)% m]=V[k];
b) for (k=1;k<=m*n;k++)  Mat[1+(k-1)/m][1+(k-1)% n]=V[k];
c) for (k=1;k<=m*n;k++)  Mat[1+(k-1)/n][1+(k-1)% n]=V[k];
d) for (k=1;k<=m*n;k++)  Mat[1+k/n][1+k%n]=V[k];
```

53. Se dă un vector **v** cu **m*n** componente numere întregi și o matrice **Mat** cu **m** linii și **n** coloane. Care dintre secvențele de mai jos copiază matricea **Mat** în vectorul **v**? Copierea se realizează astfel: mai întâi se copiază linia **1**, apoi linia **2**,... la sfârșit linia **m**.

```
a) for (i=1;i<=n;i++)
   for (j=1;j<=m;j++) V[j+(i-1)*n]=Mat[i][j];
b) for (i=1;i<=m;i++)
   for (j=1;j<=n;j++) V[j+(i-1)*n]=Mat[i][j];
c) for (i=1;i<=m;i++)
   for (j=1;j<=n;j++) V[i+(j-1)*n]=Mat[i][j];
d) for (i=1;i<=m;i++)
   for (j=1;j<=n;j++) V[j+(i-1)*n]=Mat[i,j];
```

Răspunsurile la testele grilă :

35. a) Algoritmul calculează suma numerelor obținute ca primă cifră a fiecărui număr din vector. **36. b) 37. d) 38. d) 39. b)** Algoritmul afișează primul element din vector, apoi primul dintre cele mai mici ca el, apoi primul dintre cele mai mari ca ultimul afișat etc. **40. c) 41. a) 42. d) 43. 97 100 99** Codurile caracterelor sunt date în ordinea lor alfabetică. Astfel, dacă **a** are codul **97**, **b** are codul **98**....ș.a.m.d. În absența acestei codificări, sortarea alfabetică ar fi imposibilă. **44. a)** În **Borland C++**, o variabilă neinițializată de programator, definită ca în program, este inițializată automat cu **0**. **45. b) 46. c)** Atenție! Chiar dacă inițial vectorul reține numere naturale, dacă în urma prelucrărilor va trebui să rețină numere reale, se declară de la început ca fiind cu elemente numere reale. Apoi: dacă prelucrările efectuate la un pas afectează prelucrările care se vor efectua la pașii următori, se folosește un vector auxiliar. **47. b) 48. d) 49. a)** Variabilele **i** și **j** reprezintă coordonatele colțului din stânga sus al unei submatrice. **50. d) 51. a) 52. c) 53. b)**

CAPITOLUL 6

Fișiere

6.1. Noțiunea de fișier

În practică, programele lucrează cu un volum mare de date. Stocarea acestora se face pe diverse suporturi magnetice (hard-disc, dischetă) sub formă de fișiere.

Definiție. *Se numește fișier o colecție de date omogene (adică de același tip), stocate pe suport extern și accesată printr-un nume, care reprezintă numele fișierului.*

Exemple.

1. Programele sub formă executabilă sunt memorate pe suport sub formă de fișiere. Colecția de date este dată de instrucțiunile în cod mașină care alcătuiesc programul. Se recunosc după extensia numelui care este **.exe** sau **.com**.

2. Programele în format sursă (adică textul introdus de programator) sunt stocate și ele sub formă de fișiere. Și acestea pot fi recunoscute după extensia lor - **.pas** pentru surse pascal, **.cpp** pentru surse C++. Aici colecția de date este constituită de caracterele care alcătuiesc programul.

3. Tot sub formă de fișiere sunt memorate informațiile de natură economică. De exemplu, un fișier "de materiale" reține informații asupra materialelor utilizate într-o unitate productivă. Astfel, pentru fiecare material se cunoaște numele, unitatea de masură (bucăți, kilograme), cantitatea existentă în stoc etc. Informațiile referitoare la un material constituie o înregistrare, iar ansamblul înregistrărilor constituie fișierul propriu-zis.

Șirul exemplelor ar putea continua, pentru că orice este memorat sub formă de fișier: imagini, melodii, etc.

Asupra fișierelor se pot face, sub controlul sistemului de operare sau al anumitor programe care sunt de fapt o interfață între acesta și utilizator, anumite operații elementare cum ar fi: copiere, ștergere etc și care au fost prezentate în prima parte a cărții. În acest capitol vom învăța să lucrăm cu fișiere cu ajutorul programelor făcute de noi.

În C++ se lucrează cu două tipuri de fișiere, dar noi vom studia doar primul tip.

a) Fișiere text;

b) Fișiere binare.

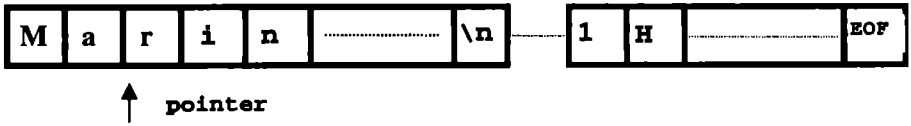
6.2. Fișiere text

6.2.1. Noțiunea de fișier text

Fișierele text se caracterizează prin următoarele:

- *datele sunt memorate sub forma unei succesiuni de caractere.*
- *Fiecare caracter este memorat prin utilizarea codului* **ASCII**. Exemple:
  - dacă dorim să memorăm caracterul **'a'**, fișierul text va reține codul **ASCII** al caracterului **'a'**.
  - în cazul în care dorim să memorăm conținutul unei variabile numerice de tip **int** care reține **123**, fișierul text va reține codurile a trei caractere: **'1'**, **'2'**, **'3'** și nu cei doi octeți necesari reprezentării în cod complementar a numărului amintit.
- *Un fișier text se termină întotdeauna cu caracterul* **EOF**. Acesta permite ca la prelucrare să poată fi identificat sfârșitul fișierului. Dacă dorim să-l introducem de la tastatură, se tastează **CTRL+Z**.
- *Fișierul text se consideră alcătuit din una sau mai multe linii.* O linie, mai puțin ultima, se termină prin caracterul **newline** (**\n**). Există cazuri în care fișierul este alcătuit dintr-o singură linie.
- *O variabilă specială, numită pointer reține întotdeauna un octet al fișierului.* Acesta este primul care va fi prelucrat - citit sau scris. După cum vedeți în figură, ne imaginăm pointerul ca o săgetuță care marchează octetul. În exemplu, aceasta marchează al treilea octet al șirului, adică cel de indice **2**.

În cele ce urmează vom prezenta schema unui fișier text.



Observații:

1. Orice editor de texte, deci și cel cu ajutorul căruia introducem textele **C++**, lucrează cu fișiere text. Astfel, programul nostru se găsește pe hard-disc sub formă de fișier text.

2. Până în prezent, am lucrat cu două tipuri speciale de fișiere text și anume cu cele ale căror nume (logice, vom vedea ce înseamnă asta) sunt **cin** și **cout**. Am spus speciale pentru că ele nu sunt memorate pe suport extern, ci corespund dispozitivelor standard de intrare, respectiv ieșire, dar, în rest, au aceleași caracteristici.

- Fișierul **cin** este privit ca fiind de intrare, adică fluxul de date este de la el către memoria internă. El este asimilat tastaturii. Mai precis, în loc ca programul să citească datele dintr-un fișier memorat pe suport, el citește aceste date din fișierul **cin**.

- Fișierul **cout** este privit ca unul de ieșire, adică fluxul datelor este de la memoria internă către el. El este asimilat monitorului. Adică în loc ca programul să scrie într-un fișier memorat pe suportul magnetic, acesta scrie pe monitor, deci în fișierul **cout**.

Toate aceste precizări au un scop. *Toate scrierile / citirile efectuate în / din aceste fișiere sunt identice cu cele efectuate în / din fișierele memorate pe suport magnetic.* Din motive pedagogice, vom analiza modul de efectuare a acestor operații în **cout / cin**, iar acestea vor fi extinse pe cele memorate pe suport extern.

3. În / din fișierele text putem scrie sau citi cu sau fără *format*. Ce înțelegem prin asta? Problema va fi tratată detaliat, aici vom da un exemplu. Să presupunem că vrem să scriem o dată reală. Dacă nu precizăm condițiile în care aceasta să fie scrisă, de exemplu, întotdeauna cu semn, chiar dacă este plus, aliniată stânga sau dreapta într-un câmp cu un număr fix de poziții, numărul de zecimale cu care să apară etc., atunci spunem că scriem fără format, altfel scriem cu format. Totuși, chiar dacă scriem fără format, se aplică anumite reguli implicite. Din acest motiv este necesar să analizăm și modul în care se efectuează operațiile respective fără format.

6.2.2 Citiri / scrieri fără format

Până în prezent am efectuat de multe ori astfel de operații. Acum le vom recapitula, prezentând în același timp anumite informații suplimentare.

➢ Citirea / scrierea variabilelor de tip **char**.

a) Citirea. Se citește primul caracter care nu este alb. După citire, pointerul indică poziția următoare caracterului citit.

b) Scrierea. Se afișează caracterul pe poziția curentă a pointerului, indiferent dacă este alb sau nu. După afișare, pointerul va indica poziția următoare.

Fie secvența:

```
char a;
cin>>a;
cout<<a;
```

- dacă tastăm **'a'** se afișează **'a'**;
- dacă tastăm **" a"** se afișează **'a'**, pentru că, la citire, au fost sărite caracterele albe - în exemplu blank-urile.

➢ Citirea / scrierea variabilelor numerice.

a) Citirea. Aceasta se face începând din poziția curentă a pointerului. Dacă numărul este precedat de semn acesta este luat în considerare. Eventualele caractere albe sunt sărite. Citirea se face până când este întâlnit un caracter care nu poate face parte din tipul respectiv (de exemplu, o literă).

b) Scrierea. Se face începând cu poziția curentă a cursorului.

1). Fie secvența:

```
int a;
cin>>a;
cout<<a;
```

Atunci:

- dacă se tastează **13,** se afișează **13**;
- dacă se tastează **bbbb14** se afișează **14** (prin **b** am notat blank).
- dacă se tastează **117abc**, se afișează **117**.

2). Fie secvența:

```
int a;double b;
cin>>a>>b;
cout<<a<<"    "<<b;
```

Atunci:

- dacă se tastează **bb17bb18.00** se afișează **17bbbb18** (observați că dacă o variabilă de un tip real reține un număr întreg, acesta este afișat fără punctul zecimal).
- dacă se tastează **112.55**, se afișează **112** (valoarea de tip **int** a fost citită până la punct) și **0.55**.

Observație: Variabilele reale se tipăresc cu cel mult 6 zecimale - dacă le au.

6.2.3. Citiri / scrieri cu format

După cum știm, *orice citire / scriere se face începând de la poziția curentă a cursorului*. Citirea / scrierea oricărui caracter determină ca acesta să sară o poziție, astfel încât următorul caracter să fie scris pe poziția următoare.

În acest paragraf ne propunem să învățăm să citim / scriem cu formatul dorit de noi. Termenii utilizați în astfel de situații sunt "scrieri / citiri cu format". Pentru aceasta este necesar să ne familiarizăm cu anumite noțiuni care au rolul de a caracteriza citirile / scrierile cu format.

1. Lățimea **-width-** se utilizează la scriere și are rolul de a stabili numărul de caractere utilizate pentru afișarea unei date.

2. Precizia **-precision-** se utilizează la scriere și se folosește atunci când se afișează o valoare reală. Are rolul de a stabili numărul de zecimale care vor fi afișate pentru valoare.

3. Caracterul de umplere **-fill-** se utilizează la scriere, în cazul în care data propriu-zisă ocupă mai puțini octeți decât lățimea, si precizează caracterul care se afișează în spațiile neocupate de dată (implicit este blank-ul).

4. Alinierea. Se utilizează în cazul în care data propriu-zisă ocupă mai puțin decât lățimea și precizează unde anume să fie afișată data - la stânga - **left-**, sau la dreapta **-right-.**

Exemple:

- Șirul **"mama"** a fost afișat cu lățimea **10** și caracterul de umplere **'x'**, aliniat dreapta:

  xxxxxxmama

- La fel ca mai sus, numai că este aliniat stânga:

  mamaxxxxxx

- Valoarea reală **0.12345** este afișată pe 8 poziții, aliniată dreapta, caracter de umplere **'f'**, cu 2 zecimale:

  ffff0.12

5. Salt sau nu peste caracterele albe - se folosește la citire.

Faceți un test. Scrieți o secvență în care se citește o dată de tip **int**. Atunci când programul așteaptă ca dvs. să introduceți data respectivă tastați înainte de a o introduce mai multe blank-uri, și **Enter**. În afara faptului că se obține o deplasare corespunzătoare pe ecran a cursorului, programul așteaptă în continuare introducerea datei respective. Aceasta înseamnă că se face salt peste caracterele albe (cele două caractere introduse sunt albe) la citire.

În cazul în care saltul peste caracterele albe este dezactivat și retestăm secvența de mai sus, vom avea eroare la citire - programul se așteaptă să citim o valoare numerică și întâlnește un caracter alb, care este nenumeric. Mai precis, în cazul în care saltul peste caracterele albe este dezactivat și este citit un prim caracter alb:

- citirea unei variabile numerice este sancționată prin eroare;
- citirea unui șir eșuează - este citit șirul vid;
- dacă se citește un caracter - este citit, de fapt, caracterul alb.

Acum vom prezenta mecanismul pus la dispoziție de limbaj prin care putem efectua citiri / scrieri cu format. Există câteva variabile și constante, definite în **iostream.h,** cu rol important în acest sens. Iată varlabilele:

```
int     x_precision;
int     x_width;
int     x_fill;
long    x_flags;
```

- **x_precision** reține numărul de zecimale ce vor fi afișate pentru un număr real;
- **x_width** reține numărul de poziții pe care se efectuează afișarea (lățimea). Spre deosebire de celelalte date membru, valoarea este reținută până în momentul în care se efectuează prima afișare, după care variabila se inițializează cu 0;
- **x_fill** reține caracterul care va fi afișat în cazul în care numărul pozițiilor pe care se face afișarea este mai mare decât cel al pozițiilor efectiv ocupate de date.
- Variabila **x_flags** reține anumiți parametri pentru formatarea intrărilor / ieșirilor. Informațiile reținute de aceasta vor fi analizate în amănunt. Parametrii sunt memorați la nivel de bit. De exemplu, dacă bitul cel mai puțin semnificativ reține 1, caracterele albe care preced valoarea sunt sărite, dacă bitul următor reține 1 datele se tipăresc aliniate stânga etc. Pentru a ușura accesul la bit s-au definit constantele:
  - **skipws=0x0001** - caracterele albe care preced valoarea care trebuie citită sunt sărite;
  - **left=0x0002** - datele se tipăresc aliniate la stânga;
  - **right=0x0004** - datele se tipăresc aliniate la dreapta;
  - **internal=0x0008** - în mod normal, valorile numerice se tipăresc prin afișarea semnului (dacă este cazul) urmat de număr. În cazul când bitul 3 (bitul 0 este cel mai puțin semnificativ) este 1 și numărul de poziții pe care se face afișarea este mai mare decât numărul de poziții efectiv ocupate de dată, semnul este afișat aliniat stânga, iar numărul aliniat dreapta. Similar se face afișarea în cazul specificatorului de bază;
  - **dec=0x0010** - conversie în zecimal;
  - **oct=0x0020** - conversie în octal;
  - **hex=0x0040** - conversie în hexazecimal;
  - **showbase=0x0080** - afișarea indicatorului de bază;
  - **showpoint=0x0100** - forțează afișarea punctului zecimal;
  - **uppercase=0x0200** - în cazul afișării în hexa, se vor utiliza literele mari: A,B....F.
  - **showpos=0x0400** - valorile numerice sunt afișate precedate de semn;

- **scientific=0x0800** - afișarea valorilor reale se face prin utilizarea formei științifice (1e-8);
- **fixed=0x1000** - afișarea valorilor reale se face prin utilizarea formei normale (-12.45);

Accesul propriu-zis la variabilele respective se face cu ajutorul unor funcții speciale, numite **manipulatori**. Pentru a le putea utiliza este necesar să includem fișierul antet **iomanip.h**.

Manipulatorii pot fi incluși în expresiile de citire/scriere ca mai jos:

cout<<manipulator$_1$<<...<<manipulator$_n$<<data
cin>>manipulator$_1$>>...>>manipulator$_n$>>variabila;

Primii trei manipulatori se utilizează exclusiv pentru scrieri:

- Stabilirea lățimii se face cu manipulatorul **setw(int)**; - unde parametrul este lățimea propriu-zisă;
- Stabilirea preciziei (numărului de zecimale) se face cu ajutorul manipulatorului **setprecision(int)**, unde parametrul reprezintă numărul zecimalelor.
- Stabilirea caracterului de umplere se face cu ajutorul manipulatorului **setfill(char);**

Analizați programul următor:

```
#include <iostream.h>
#include <iomanip.h>
main()
{
 double a=0.123456789;
 cout<<setw(10)<<setfill('x')<<setprecision(2)<<a;
}
```

Variabila **a** este afișată pe **10** poziții, cu **2** zecimale exacte, iar pozițiile rămase neocupate de către dată sunt scrise cu caracterul de umplere **'x'**.

Accesul la conținutul variabilei **x_flags** se face într-un mod specific, pentru că fiecare bit al ei are o anumită semnificație. Din acest motiv avem nevoie de accesul la biții ei. Analizați constantele utilizate la accesarea ei. Valorile lor sunt, de fapt, puteri ale lui 2. Mai mult, din analiza lor deducem și valorile pe care trebuie să le ia biții. De exemplu, dacă bitul cel mai puțin semnificativ ia valoare 1, înseamnă că se face salt peste caracterele albe. Contrar, saltul nu se face.

Accesul la biții acestei variabile se face printr-o mască - procedeu des folosit în programare. Masca precizează la care biți doresc accesul. De exemplu, doresc accesul la biții **0**, **5** - din **16** biți numerotați de la **0** la **15**. Ideea este de a pune **1** în mască pentru fiecare bit la care doresc accesul și **0** pentru biții al căror conținut rămâne nemodificat. Astfel, obțin

masca: **0000000000100001**. În acest fel accesăm biții, dar ce facem cu ei?

Sunt două operații posibile:

a) **setare** - toți vor reține 1;
b) **resetare** - toți vor reține 0.

- Manipulatorul **setiosflags(masca)** are rolul de a seta biții din mască.
- Manipulatorul **resetiosflags(masca)** are rolul de a reseta biții din mască.

În formarea măștii sunt extrem de utile constantele, care pe de o parte sunt puteri ale lui 2, pe de altă parte numele lor ne permit să înțelegem, în funcție de caz, ce biți trebuie setați.

Pentru a avea acces la constante, numele lor va fi precedat de **ios::**. Iată cum se formează masca pentru exemplul de mai sus, prin utilizarea constantelor și a operatorului "**sau**" pe biți:

```
ios::skipws | ios::oct
```

Prin setarea biților selectați prin această mască solicităm ca citirea să se facă prin saltul caracterelor albe și conversia unei valori octale.

```
setiosflags(ios::skipws | ios::oct)
```

Exemple:

1. Analizați programul următor:

```
#include <iostream.h>
#include <iomanip.h>
main()
{ int a=23;
 cin>>setiosflags(ios::skipws | ios::oct)>>a;
 cout<<a;
}
```

Se citește o valoare numerică în baza **8**. Dacă introduc **23**, programul va afișa **19** - aceeași valoare, dar în baza **10**. Dacă introducem **87** citirea se întrerupe - pentru că valoarea introdusă nu este în baza **8**. Conținutul variabilei a rămâne nemodificat.

2. Tipăresc un întreg, așa cum le place economiștilor, pe **20** de poziții, semnul la stânga și numărul la dreapta.

```
+          23


int a=23;
cout<<setw(20)<<
    setiosflags(ios::internal | ios::showpos | ios::right)<<a;
```

3. Tipăresc o valoare reală în formă științifică, așa cum nu cred că-i place cuiva: **2.3456e+01**.

```
double a=23.456;
cout<<setiosflags(ios::scientific)<<a;
```

Observații:

1. De regulă, utilizarea manipulatorilor are efect numai asupra primei citiri / scrieri, chiar dacă operația respectivă este scrisă în cadrul aceleiași expresii. Cu toate acestea există și excepții de la această regulă, de exemplu, saltul peste caracterele albe.

2. În cazul în care lățimea nu este suficientă ea este ignorată.

3. Dacă numărul de zecimale nu este precizat prin manipulator, atunci se tipăresc cel mult 6 zecimale (dacă valoarea respectivă are 6 zecimale).

4. În cazul în care nu a fost specificat caracterul de umplere, acesta este blank-ul.

5. Conversia bazelor **8, 10, 16** se poate realiza simplificat prin utilizarea manipulatorilor **oct dec hex**.

Exemplu: **cin>>hex>>a** - citesc o variabilă întreagă, iar data citită se presupune că este în hexa.

6.2.4. Fișiere text memorate pe suport magnetic

Așa cum am arătat, toate operațiile de citire / scriere efectuate asupra fișierelor cu numele logic cin / cout pot fi efectuate și asupra fișierelor text memorate pe suport magnetic. Totuși, utilizarea fișierelor text memorate pe suport magnetic impune cunoștințe suplimentare.

Pentru a putea fi prelucrat, orice fișier are două nume: numele logic și numele fizic.

- Numele logic este cel folosit în program pentru referirea fișierului.
- Numele fizic este cel sub care fișierul se găsește memorat pe suportul extern. El poate conține și calea - subdirectorul în care se găsește (altfel, fișierul se consideră plasat în directorul curent).

De la început facem precizarea că setul de comenzi cu ajutorul cărora se poate lucra cu aceste fișiere este cu mult mai cuprinzător decât cel prezentat în actualul manual.

În **C++** pentru a putea lucra ușor asupra fișierelor sunt definite anumite constante, pe care le prezentăm în continuare:

- **in** = **0x01** - fișierul se deschide pentru citire;
- **out** = **0x02** - fișierul se deschide pentru scriere;
- **ate** = **0x04** - după deschidere, salt la sfârșitul fisierului;

- **app** = **0x08** - deschidere pentru a scrie la sfârșitul fișierului;
- **trunc** = **0x10** - dacă fișierul care se deschide există, în locul său se creează altul (fișierul existent se pierde).
- **nocreate** = **0x20** - deschide fișierul doar dacă acesta există (nu este permisă crearea lui).
- **noreplace** = **0x40** - dacă fișierul există, el poate fi deschis doar pentru consultare.
- **binary** = **0x80** - fișier binar. Se utilizează constructorul implicit al clasei **ofstream()**;. Apoi se utilizează metoda **open**, în forma generală.

Orice program care lucrează cu fișiere pe suport magnetic trebuie să includă în prealabil fișierul antet **fstream.h**. În concluzie, vom scrie:

```
#include <fstream.h>
```

Includerea acestui fișier face inutilă includerea fișierului **iostream.h**.

6.2.4.1. Declararea fișierelor text memorate pe suport magnetic

Înainte de a lucra cu un fișier text, este necesar ca acesta să fie declarat. În C++ se face prin declararea lui ca fiind de un tip special, numit **fstream**. Forma generală a unei astfel de declarații este:

```
fsteam nume_logic(char* nume_fizic, int mod_de_deschidere);
```

Modul de deschidere (pentru citire, scriere, etc) se descrie cu ajutorul constantelor care au fost prezentate în subparagraful anterior și operatorul sau (|) - la fel cum se lucrează cu manipulatorii **setiosflags** sau **resetiosflags**.

Exemple:

- Declar un fișier text care se va găsi în rădăcină, cu numele fizic **fis.txt**. Fișierul a fost declarat în vederea creării lui. În eventualitatea că în rădăcină se mai găsește un fișier cu acest nume, acesta din urmă va fi distrus. Numele său logic este **f**.

```
fstream f("c:\\fis.txt",ios::out);
```

- Declar un fișier text cu numele fizic **date.txt** și numele logic **g**. Fișierul a fost declarat în vederea citirii (consultării), deci el trebuie să existe pe suport. Întrucât nu a fost precizată calea, acesta trebuie să se găsească în directorul curent.

```
fstream g("date.txt",ios::in);
```

- Declar două fișiere, unul pentru consultare, altul pentru scriere.

```
fstream f("c:\\fis.txt",ios::in),g("c:\\fis1.txt",ios::out);
```

- Am declarat un fișier pentru a scrie la sfârșitul său. De la început, pointerul va fi plasat după ultimul caracter pe care îl conține. În absența acestui parametru, pointerul va fi poziționat pe primul caracter al șirului, adică cel de indice 0.

```
fstream f("c:\\fis.txt",ios::app);
```

Observații:

1. După cum vedeți calea se declară cu două caractere *backslash* (\\). Care ar fi logica pentru ca în C++ să se folosească o altă convenție de adresare? Nici vorbă de așa ceva. Caracterul respectiv este folosit pentru a scrie secvențe escape. Din acest motiv, pentru a-l include pe el într-un șir de caractere este necesar să fie scris de două ori. De altfel, puteți face un mic test: afișați pe monitor un șir care conține acest caracter de două ori. El va apărea o singură dată. Apoi afișați un șir în care acest caracter apare o singură dată. Veți vedea că nu este afișat.

2. În anumite cazuri numele fizic al fișierului trebuie citit de la tastatură. În acest fel avem posibilitatea ca programul să prelucreze fișiere cu orice nume fizic. În acest caz declarația fișierului trebuie să conțină adresa vectorului de caractere -de fapt, numele său- și să fie plasată după citirea șirului respectiv.

```
char nume[20];
cout<<"numele fisierului ";cin>>nume;
fstream f(nume,ios::out);
```

Atenție! În astfel de cazuri, atunci când se introduce calea, este necesar să scriem un singur backslash.

După prelucrare, este necesar ca fișierul să fie închis. Închiderea unui fișier se face printr-o funcție specială `close()`, fără parametri, dar precedată de numele fișierului și punct.

Exemplu: `f.close()`. Se închide fișierul cu numele logic `f`.

6.2.4.2. Prelucrarea fișierelor text

În general, prelucrarea unui fișier se face după următoarea schemă:

```
while (daca nu este sfarsit de fisier)
  {
    citeste
    prelucreaza
  }
```

Pentru detectarea sfârșitului de fișier, C++ conține o funcție specializată numită `eof()`. Pentru a preciza fișierul al cărui sfârșit se testează, funcția trebuie precedată de numele logic al fișierului și punct. În cazul în care a fost detectat sfârșitul fișierului, funcția returnează o valoare diferită de 0,

altfel returnează 0. Totuși, trebuie mare atenție atunci când o folosim. Să vedem de ce.

Analizați secvența următoare, în care se citesc caractere din fișierul cu numele logic **f**. Ea este eronată.

```
char x;
while(!f.eof())
   {
    f>>x;
    cout<<x;
   }
```

Funcția **eof()** nu citește, ci doar testează dacă anterior a fost detectat sfârșitul de fișier. Există o variabilă unde sistemul memorează aceasta. Să presupunem că fișierul are memorat un singur caracter, de exemplu **'a'**. La prima citire se citește **'a'** și se afișează **'a'**. Sfârșitul de fișier nu a fost detectat. Se încearcă o nouă citire. De această dată se întâlnește sfârșitul de fișier și citirea eșuează, dar conținutul variabilei x rămâne nemodificat. Prin urmare, se afișează din nou **'a'**...

Cum procedăm în astfel de situații? Așa cum am văzut, la citire se folosește o expresie (...>>...). Aceasta produce o valoare diferită de **0** dacă citirea a reușit, altfel produce valoarea **0**. Acest fapt poate fi folosit în vederea unei exploatări corecte, așa cum rezultă din exemplele următoare.

1. Programul următor creează un fișier text cu intrarea de la tastatură. Caracterele citite se sfârșesc prin **EOF (CTRL+Z)** Nu sunt scrise caracterele albe introduse de la tastatură.

```
#include <fstream.h>
main()
{
fstream f("c:\\fis.txt",ios::out);
char ch;
while (cin>>ch)f<<ch;
f.close();
}
```

2. La fel, ca mai sus, numai că nu sunt sărite caracterele albe citite de la tastatură.

```
#include <fstream.h>
#include <iomanip.h>
main()
{ fstream f("c:\\fis.txt",ios::out);
  char ch;
  while (cin>>resetiosflags(ios::skipws)>>ch)f<<ch;
  f.close();
}
```

3. Afișez pe monitor fișierul creat la exemplul 2, fără caracterele albe

```
#include <fstream.h>
main()
{ fstream f("c:\\fis.txt",ios::in);
  char ch;
  while (f>>ch)cout<<ch;
  f.close();
}
```

4. Afișez pe monitor fișierul creat la exemplul 2, cu toate caracterele, inclusiv cele albe.

```
#include <fstream.h>
#include <iomanip.h>
main()
{fstream f("c:\\fis.txt",ios::in);
 char ch;
 while (f>>resetiosflags(ios::skipws)>>ch)cout<<ch;
 f.close();
}
```

5. Scriu la sfârșitul fișierului creat la exemplul 2 alte caractere, inclusiv cele albe citite de la tastatură.

```
#include <fstream.h>
#include <iomanip.h>
main()
{ fstream f("c:\\fis.txt",ios::app);
 char ch;
 while (cin>>resetiosflags(ios::skipws)>>ch) f<<ch;
 f.close();
}
```

6. Conținutul fișierului **"fis.txt"** este copiat în fișierul **"fis1.txt"**. Sunt copiate și caracterele albe.

```
#include <fstream.h>
#include <iomanip.h>
main()
{
  fstream f("c:\\fis.txt",ios::in),g("c:\\fis1.txt",ios::out);
  char ch;
  while (f>>resetiosflags(ios::skipws)>>ch)g<<ch;
  f.close();
}
```

7. Programul de mai jos creează un fișier text cu primele **100** de numere naturale. Acestea sunt memorate în hexa. Fiecare număr este scris pe o linie, ocupă 5 caractere și este aliniat dreapta (implicit).

```
#include <fstream.h>
#include <iomanip.h>
main()
{ fstream f("c:\\numere.in",ios::out);
  int i;
```

```
    for (i=1;i<=100;i++)
        f<<setw(5)<<hex<<i<<endl;
   f.close();
}
```

Testul sfârșitului de fișier, în cazul citirii unor variabile numerice se face la fel ca în cazul caracterelor.

8. Programul următor listează pe monitor fișierul creat la exemplul 7. Citirea se face în hexa, dar listarea se face în baza **10**.

```
#include <fstream.h>
#include <iomanip.h>
main()
{ fstream f("c:\\numere.in",ios::in);
  int i;
  while (f>>hex>>i)cout<<dec<<i<<endl;
  f.close();
}
```

Bine, veți întreba, dar dacă nici în cazul citirii caracterelor, nici în cazul valorilor numerice funcția **eof()** nu ne este de ajutor, atunci care este rostul ei? Ea este de folos în cazul citirii șirurilor de caractere, dar șirurile de caractere le veți studia anul viitor.

9. În cazul în care se deschide un fișier care există pentru creare, vechiul fișier este distrus. În acest fel, din neatenție, se pot pierde date importante. Pentru a evita aceasta se poate recurge la următorul artificiu:

- Încercăm să deschidem fișierul cu același nume fizic, dar pentru citire (**ios::in**);
- Se testează dacă fișierul a fost deschis. În cazul în care operația a reușit, numele logic al acestuia returnează o valoare diferită de 0, altfel returnează 0. În prima situație, programul va da un mesaj -de exemplu, "fișier existent" și este oprită execuția sa, altfel vom deschide fișierul cu un alt nume logic dar același nume fizic pentru ieșire. Un program poate fi oprit la întâlnirea instrucțiunii **return 0**. Această instrucțiune nu a fost prezentată, o folosim fără alte explicații.

```
#include <fstream.h>
#include <iomanip.h>
main()
{
 fstream f("c:\\fis.txt",ios::in), g("c:\\fis.txt",ios::out);
 char ch;
 if (f)
  {  cout<<"fisierul exista";
     f.close();
     return 0;
  }
 while (cin>>resetiosflags(ios::skipws)>>ch) g<<ch;
 f.close();
}
```

10. Programul următor citește un fișier text alcătuit din mai multe linii - unele pot fi și vide. El afișează liniile fișierului citit. Observați că și în cazul fișierelor putem folosi fără probleme funcția **get**.

```
#include <fstream.h>
main()
{
 fstream f("c:\\fis.txt",ios::in);
 char linie [100];

while (f.get(linie,100))
  {
    cout<<linie<<endl;
    f.get();
  }
}
```

Programatorul poate acționa asupra pointerului dintr-un fișier de intrare. Există mai multe funcții care ne sunt de folos în acest caz.

- Funcția **long tellp()** returnează poziția pointerului la un moment dat. Poziția este relativă la începutul fișierului.
- Funcția **seekp(long, seek_dir)** poziționează pointerul. Primul parametru reprezintă poziția, iar al doilea reperul în raport de care este calculată poziția. În acest sens, au fost definite 3 constante. Pentru accesul la ele se folosește și de această dată **ios::**.
  - **beg** - început de fișier;
  - **cur** - poziția curentă în fișier;
  - **end** - sfârșit de fișier.

Evident, când stabilim o poziție aflată în stânga reperului; primul parametru este negativ, iar în dreapta reperului, parametrul este pozitiv.

11. Fiind dat un fișier text **fis.text** și două numere naturale **i<j** se cere să se listeze caracterele fișierului text dintre **i** și **j** (inclusiv) în alt fișier text numit **fis1.txt**.

```
#include <fstream.h>
main()
{ fstream f("c:\\fis.txt",ios::in);
  int i,j;
  char ch;
  cout<<"i=";cin>>i;
  cout<<"j=";cin>>j;
//pozitionez pointerul direct pe octetul i
  f.seekp(i,ios::beg);
// atat timp cat poniterul este mai mic sau egal cu j
  while (f.tellp()<=j)
     { f>>ch;
       cout<<ch;
     }
 f.close();
}
```

6.2.5. Aplicații cu fișiere text

O serie de aplicații de mare utilitate a fișierelor test se referă la testarea programelor. La ce mă refer? De regulă, când scriem un program, îl testăm cu câteva date introduse de la tastatură. Astfel de teste nu sunt semnificative. S-ar putea ca programul să aibă *bug*-uri (termen folosit de programatori, se referă la erorile ascunse care le-ar putea avea un program). Apoi, nu vedem "pe viu" cât de eficient este programul nostru, deși anumite calcule de eficiență se pot face înainte de a scrie programul. Din acest motiv datele de test se pot genera aleator, în număr mare și reținute în fișiere text. Programul nostru va citi atât datele de intrare din fișiere text, iar datele de ieșire vor fi scrise tot în fișiere text.

Atenție. La toate concursurile de informatică programele concurenților sunt testate cu ajutorul fișierelor text.

1. *Să se genereze un fișier text cu* **n** *numere naturale de tip* **int**. *Numărul* **n** *este citit de la tastatură. Numerele vor fi generate aleator. Testați programul pentru* **n>10000**.

```
#include <fstream.h>
#include <stdlib.h>
main()
{  fstream f("fis1.txt",ios::out);
   long n,i;
   cout<<"n=";cin>>n;
   f<<n<<endl;
   randomize();
   for (i=0;i<n;i++) f<<rand()<<' ';
   f.close();
}
```

2. *Să se sorteze crescător numerele din fișierul text anterior creat. Datele sortate vor fi scrise în alt fișier text.*

```
#include <fstream.h>
int v[20000],n,i,man,gasit
main()
{ fstream f( "fis1,txt",ios::in),
           g("fis2.txt",ios::out);
  //citesc datele
  f>>n;
  for (i=0;i<n;i++) f>>v[i];
  // sortez
  do
  {  gasit=0;
    for(i=0;i<n-1;i++)
    if (v[i]>v[i+1])
     {  man=v[i];
        v[i]=v[i+1];
        v[i+1]=man;
        gasit=1;
     }
  }while(gasit);
```

```
//tiparesc
 for(i=0;i<n;i++)g<<v[i]<<' ';
}
```

Observații:

1. Algoritmul de sortare este cel clasic, diferă numai intrarea și ieșirea.

2. Pentru **n** suficient de mare (**n>30000**) programul nu va funcționa. Motivul? Este insuficientă memoria internă alocată programului. Atunci? Iar intervin algoritmii fundamentali. Am putea "sparge" vectorul cu multe numere în doi sau mai mulți vectori. Apoi, sortăm pe fiecare dintre ei. Soluția finală, va fi obținută prin interclasarea vectorilor deja sortați. Dar interclasarea nu solicită multă memorie internă? Nu, după cum rezultă din analiza programului următor, unde se interclasează numerele aflate în două fișiere text.

3. *Să se interclaseze datele aflate în două fișiere text. Rezultatul va fi scris în alt fișier text.*

```
#include <fstream.h>
main()
{
 long a,b,i,j,m,n,flag;
 fstream f("fis1.txt',ios::in),
  g("fis2.txt",ios::in),
  h("fis3.txt",ios::out);
  i=j=1;
  f>>m>>a; g>>n>>b;h<<n+m<<endl;
  while (i<=m && j<=n)
   if (a<=b)
    {
     h<<a<<' ';
     f>>a; i++;
     flag=1;
    }
   else
    {
     h<<b<<' ';
     g>>b;  j++;
     flag=2;
    }
if (flag==1)h<<b<<' ';
 else h<<a;
while (f>>a)h<<a<<' ';
while (g>>b)h<<b<<' ';
f.close(); g.close(); h.close();
}
```

4. Programul următor "sparge" un fișier text cu numere generate aleator, în alte două fișiere text. El poate fi folosit cu succes în sortarea unui număr foarte mare de date întregi.

```
#include <fstream.h>
main()
{ long m,n,i,j,man;
 cout<<" fisierul de intrare ";cin>>intr;
```

```
cout<<" prima iesire ";cin>>ies1;
cout<<" a doua iesire  ";cin>>ies2;
fstream f("fis1.txt",ios::in),
 g("fis2.txt",ios::out),
 h("fis3.txt",ios::out);
f>>n;
m=n/2;
n-=m;
g<<m<<endl;
for (i=0;i<m;i++)
 { f>>man;
   g<<man<<' ';
 }
 h<<n<<endl;
for(i=0;i<n;i++)
 {f>>man;
  h<<man<<' ';
 }
f.close();g.close(),h.close();
}
```

6.2.6 Alte posibilități de citire

O întrebare des pusă este: *cum se depistează sfârșitul de linie pentru un fișier text?* Pentru aceasta se utilizează o funcție specifică:

```
getline(char* Adresa_sir, int Nr_car, char='\n')
```

Funcția citește un șir de caractere până când una dintre condițiile următoare este îndeplinită:

a) Au fost citite **Nr_car-1** caractere;
b) A fost detectat sfârșitul de linie: **'\n'**.

Evident, în acest fel, la o citire pot fi aduse în memorie cel mult **32766** caractere (vezi limitele tipului **int**).

✓ Caracterul care marchează sfârșitul liniei **'\n'** nu este inserat în șir, în schimb este inserat caracterul nul (marchează sfârștul șirului);

✓ Este suficient să apelăm funcția cu primii doi parametri, ultimul este implicit;

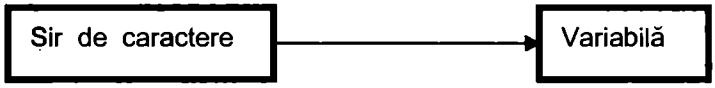
✓ Dacă se dorește, ultimul parametru poate fi apelat explicit cu o valoare convenabilă, caz în care citirea se face până la întâlnirea ei sau până au fost citite **Nr_car-1** caractere.

1. Programul următor citește linie cu linie un fișier text ale cărui linii nu au mai mult de **500** de caractere și afișează liniile pe monitor. Nu se cunoaște lungimea fiecărei linii.

```
#include <fstream.h>
main()
{  fstream f("f.dat",ios::in);
   char Linie_citita[501];
   while (f.getline(Linie_citita,501))
     cout<<Linie_citita<<endl;
}
```

Există și alte posibilități a de prelucra șirurile de caractere.

Un tip special, numit **istrstream,** permite chiar declararea stream-urilor de la șiruri către variabile. "Citirea" se efectuează cu ajutorul operatorului >> la fel ca din fișier.



2. Priviți programul următor.

1. Un șir, **X**, reține numerele **1 2 3 4 5**. O funcție specială (numită constructor) atașează șirului **X**, un stream, numit **ins**. Ea are doi parametri: adresa **X** și lungimea lui: **istrstream ins(X, strlen(X));**
2. "Citirea" se efectuează așa cum suntem obișnuiți. După fiecare citire se incrementează variabila **i**, care la sfârșit este afișată.

```
#include <iostream.h>
#include <strstrea.h>
#include <string.h>
main()
{  char X[]="1 2 3 4 5";
   istrstream ins(X, strlen(X));
   int nr;
   while (ins>>nr) cout<<nr<<endl;
}
```

✓ Observați cât de simplu se detectează sfârșitul șirului.

3. Se dă un fișier text care conține mai multe linii. Fiecare linie conține un număr oarecare de numere naturale. Se cere să se scrie un program care citește toate liniile fișierului. Fiecare linie citită va fi afișată. De asemenea, pentru fiecare linie citită se va afișa numărul de numere pe care le conține.

Fișierul de test a fost creat chiar din editorul **Borland C++.** Observați că el conține o linie vidă.

```
c:\borlandc\bin\f.dat
1 2 3 4 5 6 7 8 9
2 3 4 5 6 7

2 3 4 5 6 7
2 1 2 3 3 4 5 65
1 2 3 8 6 5
```

Iată programul și rezultatele afișate:

```
#include <fstream.h>
#include <strstrea.h>
#include <string.h>
main()
{  fstream f("f.dat",ios::in);
   char Linie_citita[501]; int nr,numere;
   while (f.getline(Linie_citita,501))
   {
      cout<<Linie_citita<<endl;
      istrstream ins(Linie_citita,
       strlen(Linie_citita));
      nr=0;
      while (ins>>numere) nr++;
      cout<<nr<<endl;
   }
}
```

```
1 2 3 4 5 6 7 8 9
9
2 3 4 5 6 7
6

0
2 3 4 5 6 7
6
2 1 2 3 3 4 5 65
8
1 2 3 8 6 5
6
```

6.3 O altă modalitate de citire scriere

Până în prezent am utilizat o modalitate de citire/scriere existentă în limbajul C++. În limbajul C clasic se utilizează și o altă modalitate de citire / scriere, modalitate care va fi prezentată în continuare. Începeți prin a include fișerul **stdio.h**: **#include<stdio.h>**

➔ Pentru a scrie pe monitor se utilizează funcția **printf**, care are forma generală:

int printf("sir", expresie$_1$, expresie$_2$,...)

Șirul este o succesiune de caractere ca mai jos:

[secv$_1$] [specificator de format] [secv$_2$] [specificator de format]...

unde:

- secv$_i$ este o secvență oarecare de caractere;
- specificator de format are o sintaxă precisă și are rolul de a stabili modul de conversie a expresiei care îi corespunde, spre a fi scrisă.

Principiul de executare. Inițial, pointer-ul indică primul caracter din șirul transmis ca parametru. Avem următoarele posibilități:

A) Este întâlnit un caracter, altul decât **%**. În principiu, acest caracter este scris. De exemplu, dacă se citește caracterul a, acesta este scris. Există posibilitatea să se întâlnească anumite caractere, cu rol special:

\n - (newline) cursorul sare pe linia următoare;

\a - (bell) calculatorul emite un sunet;

\b - (backspace) cursorul se mută pe poziția anterioară și caracterul care era acolo este șters;

\r - (CR) cursorul sare pe linia următoare;

\t - (HT tab orizontal) cursorul sare în dreapta cu **5** poziții;

B) Este întâlnit caracterul **%**. Se verifică dacă succesiunea caracterelor următoare constituie un specificator de format.

În cazul în care sintaxa este îndeplinită, specificatorul de format este asociat primei expresii, aceasta este converită și scrisă conform cerinței, apoi, în șirul transmis ca parametru pointer-ul va indica primul caracter care nu corespunde specificatorului de format. În cazul în care sintaxa nu este îndeplinită se tipărește caracterul procent, iar pointer-ul va indica caracterul următor.

Un specificator de format are rolul de a stabili modul în care se tipărește expresia care îi corespunde. Forma sa este:

% [aliniere semn][latime][.precizie] tip conversie;

Să presupunem că dorim să tipărim conținutul unei variabile. Pentru tipărire se tipăresc **n** poziții, dar valoarea care se tipărește ocupă **p<n** poziții.

Există două moduri de aliniere:

- la dreapta (implicit): ultimele p poziții din cele n sunt ocupate de valoarea tipărită;
- la stânga: primele **p** poziții din cele **n** sunt folosite pentru tipărire, restul rămân libere.

Specificațiile părților care alcătuiesc un specificator de format sunt:

- aliniere semn, dacă este prezent poate lua una din valorile:

'-' alinierea se face la stânga, iar în dreapta se completează cu spații.

'+' este trecut semnul valorii numerice (implicit semnul apare numai pentru numerele negative).

' ' dacă valoarea este pozitivă în locul semnului este trecut spațiu, altfel este trecut semnul -.

- lățime - are rolul de a preciza numărul de poziții pe ecran în care se afișează data respectivă. Dacă aceasta ocupă mai mult, parametrul este ignorat. Dacă ea ocupă mai puține caractere, se procedează conform parametrului aliniere semn. Dacă acest parametru este absent se afișează din poziția curentă a cursorului, pe numărul de poziții ocupate de dată.

- precizie - are rolul de a defini precizia cu care se tipărește data în cadrul celor n poziții (precizate de lățime). Semnificația este diferită în funcție de natura datei care se tipărește. Eventuala poziție ocupată de semn nu este numărată.
  - Pentru datele de unul din tipurile întregi stabilește numărul pozițiilor pe care se face tipărirea propriu-zisă. Dacă data ocupă mai mult parametrul este ignorat. Dacă ocupă mai puțin se tipărește precedată de un număr corespunzător de cifre **0**.
  - pentru datele din unul din tipurile reale precizează numărul de zecimale care se tipăresc (implicit acest număr este **6**).
- tip conversie - o dată care urmează a fi tipărită are un anumit tip. (**int**, **float** etc). Ea trebuie să apară pe monitor în forma dorită de programator. Prin urmare, este necesară conversia ei. Tipul conversiei rezultă de mai jos:

Caracter	**Tip**	**Cum apare pe ecran**
d	întreg	întreg cu semn;
o	întreg	octal fără semn;
u	întreg	întreg fără semn;
x	întreg	hexa fără semn, litere mici;
X	întreg	hexa fără semn, litere mari;
f	real	valoare cu semn;
x	întreg	hexa fără semn, litere mici
c	char	caracter
s	adresă șir de caractere	tipărește caracterele șirului.

Exemple:

1. Programul de mai jos afișează: **valoarea lui n este 12**

```
#include <stdio.h>
main()
{  int n=12;
   printf(" valoarea lui n este %d", n); }
```

2. Fie declarația: **int a=-12;** Atunci:

- **printf(" a=%d\n", a);** afișează **a=12**, iar cursorul sare pe rândul următor;
- **printf(" a=%10.7d", a);** afișează **a=bb-0000012**;

3. Fie declarațiile: **char a=5, b='m';**

printf("numar %d, caracter %c, cod caracter %d", a,b,b); afișează: **numar 5, caracter m, cod caracter 109.**

```
4. float a=67.68; double b=-98999999.912;
   printf ("a=%.3f\nb=%.3f", a,b);
```

Afișează:

```
a=67.680
b=98999999.912;
```

✓ Funcția **printf** întoarce ca rezultat numărul de octeți scriși.

➔ Citirea datelor se face cu funcția **scanf**

int scanf("sir", adresă variabilă$_1$, adresă variabilă$_2$...)

Parametrul **sir** are forma următoare:

[secv$_i$][specificator de format$_i$][secv$_2$][specificator de format$_i$]...

- **secv$_i$** este o secvență oarecare de caractere. În mulțimea caracterelor care o alcătuiesc un rol aparte îl joacă caracterele albe (whitespaces). Acestea sunt: blank (' '), tab orizontal (**\t**), tab vertical (**\v**), new line (**\n**), cr (**\r**). După cum vom vedea, ele se tratează într-un mod special.
- specificator de format are rolul de stabili modul în care șirul de intrare este convertit pentru a fi memorat.
- **adresă variabilă$_i$** este adresa variabilei **i** și se obține scriind caracterul **&** înaintea numelui de variabilă. De exemplu, dacă variabila se numește **a**, scriem **&a**.

specificator de format$_i$ precizează modul în care se interpretează datele și are forma generală:

%[*]latime]tip_conversie nu lipseste o paranteza?

unde:

% marchează începutul unui specificator de format;

***** dacă este prezent, câmpul se citește, dar nu este memorat și nici numărat (observația se referă la valoarea întoarsă de **scanf**).

lățime reprezintă numărul maxim de octeți din câmpul de intrare care vor fi citiți.

tip_conversie precizează natura conversiei:

caracter	ce se așteaptă	adresă variabilă de tip
d	întreg în baza **10**	**int**
D	întreg în baza **10**	**long**
f	real	**float**
lf	real	**double**
o	întreg în baza **8**	**int**
u	întreg fără semn	**unsigned int**
x	întreg în baza **16**	**unsigned int**
s	șir de caractere	**char[]**

Principiul de executare pentru **scanf**: se analizează două șiruri de caractere:

1. Șirul de caractere transmis ca parametru funcției;
2. Șirul de caractere care constituie intrarea.

În funcție de natura caracterului indicat de șirul parametru avem 3 situații:

A Caracter oarecare (nu este alb sau **%**);
B Caracter alb;
C Caracterul este **%.**

A. Se testează existența în șirul de intrare a caracterului respectiv. Avem două posibilități: dacă în șirul de intrare se găsește caracterul respectiv se avansează în ambele șiruri cu câte un caracter, iar dacă acesta nu este găsit citirea se întrerupe și funcția returnează numărul de câmpuri citite până în acel moment.

B. Pointer-ul indică în șirul parametru un caracter alb. Se analizează caracterul din șirul de intrare: dacă acesta este găsit se sare peste aceesta și peste primele caractere albe care urmează și în șirul de intrare pointer-ul va indica primul caracter care nu este alb. În caz contrar (dacă în șirul de intrare nu se găsește caracter alb), atunci pointer-ul în șirul de intrare rămâne nemodificat, iar în șirul parametru pointer-ul va indica primul caracter care nu este alb.

C. Se verifică dacă secvența următoare îndeplinește condițiile de sintaxă ale unui specificator de format. Dacă nu sunt îndeplinite condițiile, caracterul se tratează ca unul oarecare, altfel se efectuează citirea conform specificatorului de format. Dacă avem de citit un caracter se citește caracterul și se memorează codul său (indiferent dacă este alb sau nu). Dacă avem de citit o valoare numerică, atunci se sar eventualele caractere albe și se citește din șirul de intrare până când se întâlnește un caracter care nu poate fi convertit către tipul respectiv sau au fost citite un număr de caractere egal cu cel specificat de parametrul lățime.

✓ Punctul care indică prezența zecimalelor este un caracter care se numără.

✓ Dacă nu a fost citit și convertit nici un caracter citirea se întrerupe.

Exemple:

1. Se citește **a** și **x**. Pentru a se introduce valoarea **3**, iar pentru **x 1.234**.

```
#include <stdio.h>
main()
{ int a; double x;
 scanf ("%d %lf", &a, &x);
 printf ("a=%d\nx=%.2f",a,x);
}
```

Se afișează:
a=3;
b=1.23.

2. Se citește un caracter și se afișează caracterul citit și codul său.

```
char car;
scanf ("%c", &car);
printf ("caracterul este %c si are codul %d",car,car);
```

3. Se afișează prima cifră a unui număr natural. Se introduce, de exemplu, **123** și se afișează **1**.

```
int cifra;
scanf ("%1d", &cifra);
printf ("%d ",cifre);
```

4. Se introduc **3** și **5**, și se afișează **5**. Practic, primul număr introdus este sărit.

```
int a;
scanf ("%*d %d", &a);
printf ("%d",a);
```

5. Se introduce **123zvbx**. Se afișează **123**.

```
int a;
scanf ("%d", &a);
printf ("%d",a);
```

6. Dacă se introduce **xbbbby (b=blank)** se afișează **x** și **y** (cele două formate de citire sunt separate printr-un spațiu). Dacă cele două formate n-ar fi fost separate prin spațiu, s-ar fi afișat **x** și... spațiu.

```
char a, b;
scanf ("%c %c", &a, &b);
printf ("%c %c",a, b);
```

Probleme propuse

1. Să se creeze un fișier text care conține linii cu un număr mic de 65 de caractere. Fișierul se va găsi pe unitatea **A:**.

2. Să se afișeze fișierul text creat anterior. După afișarea a **20** de linii, programul așteaptă apăsarea unei taste. Aceasta se realizează prin citirea unui caracter de orice tip.

3. Să se creeze un fișier text care pe prima linie reține numărul **n<25000**, **n** citit de la tastatură, iar pe a doua **n** numere naturale generate aleator, separate prin blank-uri.

4. Să se calculeze valoarea minimă reținută de fișierul precedent.

5. Să se creeze un fișier text astfel:

- prima linie îl conține pe **n**, număr natural, **n<50000**;
- următoarele **n** linii conțin fiecare două valori întregi **0<p,q≤32000**, generate aleator.

6. Presupunând că fișierul de la 5 reține **n** numere raționale unde **p** este numărătorul și **q** numitorul, se cere să se creeze un fișier cu aceeași organizare a informațiilor, dar în care numerele raționale sunt sortate descrescător.

7. Se citesc două numere naturale **0<a,b≤32000** care reprezintă numărul rațional **a/b**. Să se scrie un program care decide dacă acest număr rațional se găsește în fișierul creat la **6**. Se va utiliza algoritmul de căutare binară.

8. Prin permutarea primelor **n** numere naturale se înțelege șirul obținut cu toate aceste numere, în care fiecare număr apare o singură dată și numerele sunt scrise într-o ordine oarecare. Exemplu: dacă **n=3**, o permutare a primelor **3** numere naturale este **312**. Se citește de la tastatură **n** număr natural, **n<25000**. Să se scrie un program care generează aleator o permutare pe care o scrie într-un fișier text, pe o singură linie. În plus, programul va scrie numerele fără blank-uri între ele. Numărul **n** nu este trecut în fișier. Pentru exemplul dat, programul scrie **321**.

Indicație. Pentru generarea permutării se pornește de la un vector care reține, în ordine, primele **n** numere naturale. Apoi, pe parcursul unui interval de timp care se citește de la tastatură, se fac inversări ale elementelor de pe pozițiile **p** și **q** cu **0<p, q≤n, p, q** generate aleator. În final, vectorul este scris în fișier.

9. Scrieți un program care citește un fișier ca cel de la problema **8** și care afișează numărul **n** de la care s-a pornit în generarea permutării.

10. Să se creeze un fișier text la fel ca la problema **8**, dar numerele sunt scrise cu spații între ele si lipsește unul singur.

11. Se citește fișierul de la exemplul **10**. Care număr lipsește? Cerință suplimentară: se va face o singură parcurgere a fișierului.

12. Toți iepurii dintr-o lovitură... Mai mulți "iepurași" se plimbă pe culoare orizontale, fiecare pe câte un culoar. Culoarele sunt împărțite într-un număr impar de cămărute și fiecare din ele are un număr dat de cămărute. Cămărutele din mijloc sunt așezate pe o dreaptă verticală. Pe această dreaptă este așezat un vânător. Acesta poate împușca orice iepuraș care se găsește în cămăruța din mijloc a culoarului său. Se presupune că un glonț poate trece printr-un număr oricât de mare de iepurași.

problemă propusă de autor, Lugoj 97.

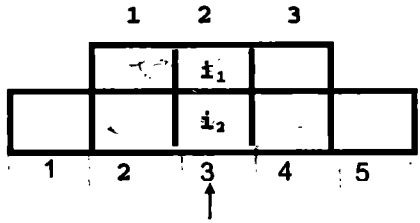
Fișierul text **iepuri.txt** conține:
Linia 1: **n** - numărul de culoare;
Următoarele **n** linii conțin informații pentru un culoar:

- l - numărul de camere;
- **c** - cămăruța în care se găsește inițial iepurașul (pentru fiecare culoar, camerele se numerotează de la stânga la dreapta între **1** și l).

Se știe că fiecare iepuraș pleacă inițial către cămăruța din dreapta culoarului său, apoi își continuă drumul către cea din stânga, iar către cea din dreapta, etc. În fiecare unitate de timp oricare din iepurași trec dintr-o cameră în alta.

Cât timp trebuie să aștepte vânătorul pentru ca să poată împușca toți iepurașii utilizând un singur glonț (dacă acest fapt este posibil)? Presupunem că vânătorul a ratat momentul (nu a tras). Cât timp trebuie să aștepte pentru a fi iar în situația de a împușca toți iepurii cu un singur glonț? Datele se afișează pe monitor.

Exemplu: **n=2, l_1=3, c_1=2 l_2=5, c_2=3**



16. Se dau două funcții f, g: [a,b)→ℜ care au expresiile:

$$f(x)=\begin{cases} m_1x+n_1 & x\in[a,t_1) \\ m_2x+n_2 & x\in[t_1,t_2) \\ \\ \\ m_rx+n_r & x\in[t_{r-1},b) \end{cases} \qquad g(x)=\begin{cases} p_1x+q_1 & x\in[a,z_1) \\ p_2x+q_2 & x\in[z_1,z_2) \\ \\ \\ p_sx+q_s & x\in[z_{s-1},b) \end{cases}$$

$$Unde \quad m_1...m_r, n_1...n_r, p_1...p_s, q_1...q_s, a, b, t_1...t_{r-1}, z_1...z_{s-1} \in \Re$$

A) Pentru o valoare reală x_0, citită de la tastatură, se cere să se afișeze: **f(x0)** și **g(f(x0))**. Pentru fiecare calcul care nu se poate efectua din considerente matematice se afișează pe monitor mesajul "**invata omule matematica**".

B) Să se rezolve ecuația **f(x)=g(x)**. Rădăcina (rădăcinile) se afișează pe monitor cu 3 zecimale. În cazul în care ecuația nu are soluție se afișează pe monitor mesajul "**ecuatia nu are solutie'.**

Datele de intrare se citesc din fișierul text **f.in** și sunt aranjate pe linii astfel:

```
r
m1 n1 a t1
m2 n2 t1 t2
. . .
mr nr tr-1 b
s
p1 q1 a z1
p2 q2 z1 z2
. . .
ps qs zs-1 b
```

CAPITOLUL 7

Complexitatea algoritmilor

7.1 Exprimarea complexității

Fiind dat un anumit algoritm, se pune problema să găsim un indicator care să exprime complexitatea sa. Acest indicator va face abstracție de calculatorul pe care rulează programul obținut în urma cuantificării algoritmului într-un limbaj de programare, precum și de limbajul de programare ales. Altfel spus, indicatorul va exprima complexitatea unui algoritm care poate fi redactat în pseudocod, schemă logică sau limbaj de programare.

⇒ Presupunem că algoritmul are **n** date de intrare și acestea urmează să fie prelucrate. Indicatorul care exprimă complexitatea va ține cont de acest număr.

Exemple:

a) Se cere să se calculeze valoarea maximă a unui șir de **n** numere reale;
b) Să se determine dacă **n** numere reale sunt distincte. Se va compara primul număr cu toate celelalte, al doilea cu cele care îi urmează ș.a.m.d.

⇒ Un algoritm conține mai multe operații de atribuire, decizionale, scriere și citire. Acestea se execută de un număr de ori care depinde de **n**, dar și de datele propriu-zise. Teoretic, ar trebui să se determine de câte ori se execută fiecare operație. Cum această cerință este imposibil de realizat în practică, s-a preferat o altă metodă. Mai precis, *se alege o operație de bază, se determină de câte ori se execută aceasta și se presupune că restul operațiilor se execută de un număr de ori care este proportional cu numărul de executări a operației de bază.* De regulă, operația de bază aleasă este operația decizională (**if**).

Exemple:

a) Pentru a calcula valoarea maximă a unui șir de **n** numere reale se efectuează **n-1** comparații.
b) Pentru a determina, prin algoritmul de mai sus dacă **n** numere reale sunt distincte, se efectuează:

$$(n-1)+(n-2)+...1=\frac{n(n-1)}{2}$$ comparații.

⇒ Exprimarea complexității unui algoritm se face aproximativ. În cazul în care expresia care cuantifică numărul de operații de bază este sub formă de polinom, ca în exemplele date, se precizează numai primul termen al acestuia, fără coeficient. Pentru a preciza că este vorba de o cuantificare aproximativă vom nota complexitatea prin **O(f(n))**.

Exemple:

a) Algoritmul care calculează maximul a **n** numere reale are complexitatea **O(n)**, deși se efectuează **n-1** comparații.

b) Algoritmul care decide dacă **n** numere reale sunt distincte are complexitatea $O(n^2)$, deși se efectuează cel mult $\frac{n(n-1)}{2}$ comparații.

✓ Algoritmii de complexitate **O(n)** se mai numesc și *algoritmi liniari*.

⇒ Există algoritmi care au *complexitate exponențială*, adică ea este de forma $O(a^n)$.

Exemplu: se cere să se afișeze toate submulțimile unei mulțimi **{1,2,...n}**. Revedeți algoritmul din manualul pentru clasa a **IX**-a. Avem 2^n submulțimi ale unei mulțimi de numere reale. Dacă considerăm ca operație principală afișarea unei submulțimi, înseamnă că algoritmul are complexitatea $O(2^n)$.

✓ În practică trebuie evitați algoritmii care nu au complexitate polinomială. Imaginați-vă, că pentru un **n** dat, un calculator performant rulează programul într-o oră. Dacă mărim pe **n** cu **10**, avem 2^{n+10} operații elementare, adică 1024×2^n operații. Prin urmare, programul va rula pe același calculator în **1024** de ore. Dar dacă mărim pe **n** cu **1000**? Practic, *pentru valori nu prea mari ale lui n, algoritmii exponențiali sunt de neutilizat.*

✓ Există și algoritmi care sunt asimilați algoritmilor exponențiali. Exemplu: se cere să se afișeze toate permutările mulțimii: **{1,2,...n}**. Avem **n!** permutări. Dacă considerăm ca operație de bază afișarea unei permutări, avem **n!** operații elementare, deci algoritmul are complexitatea **O(n!)**. Dar $n! = 1 \times 2 \times 3 \times \ldots n > 1 \times 2 \times 2 \times \ldots \times 2 = 2^{n-1}$.

Până în prezent am văzut că algoritmii au complexitate $O(n^k)$ (polinomiali) și $O(a^n)$ (exponențiali).

⇒ Există și algoritmi care au complexități de forma **O(log(n))**, **O(n×log(n))**, $O(n^2 \times \log(n))$, unde prin **log(n)** înțelegem $\log_2(n)$.

✓ Deocamdată nu ați studiat la matematică logaritmii. Rețineți:

$$2^n = x \Leftrightarrow n = \log_2(x)$$

Cu alte cuvinte, prin logaritm în baza **2** din **x** ($\log_2(x)$) înțelegem puterea la care trebuie ridicat numărul **2** astfel încât $2^n=x$. Veți învăța că logaritmul în baza **2** din **x** este mult mai mic decât **x**. De exemplu, $\log_2(1024) = 10 < 1024$. De asemenea, avem identitatea: $\log_2(x^n) = n \cdot \log_2(x)$.

Exemplu: Căutarea binară. Se citește un vector cu **n** componente numere întregi, unde numerele se presupun ordonate crescător și o valoare întreagă (**nr**). Să se decidă dacă **nr** se găsește sau nu printre numerele citite, iar în caz afirmativ să se tipărească indicele componentei care conține acea valoare.

O rezolvare în care **nr** se compară pe rând cu cele **n** valori, este lipsită de valoare (nu exploatează faptul că cele **n** valori sunt în secvență crescătoare). Algoritmul care va fi propus este mult mai performant și face parte, așa cum am învățat, dintre algoritmii clasici.

O metodă de estimare a numărului de comparații făcute de algoritm, este dată în continuare.

Notez cu **T(n)** numărul de comparații efectuate de algoritmul căutării binare pentru un vector cu **n** componente. Pentru simplitate, vom considera $n=2^k$.

La un pas se compară elementul căutat cu cel din mijloc - de indice: $\left[\frac{i+j}{2}\right]$

Se efectuează o comparație după care problema se reduce la alta mai simplă:

Astfel, avem:
$$T(n) = \begin{cases} 1, & \text{daca } n = 1; \\ T\left(\frac{n}{2}\right) + 1, & \text{altfel.} \end{cases}$$

$$T(n) = T\left(2^k\right) = 1 + T\left(2^{k-1}\right) = 2 + T\left(2^{k-2}\right) = \ldots k = \log_2 n$$

Dacă **n** nu este de forma 2^k, atunci există **k** astfel încât:

$$2^k < n < 2^{k+1} \Rightarrow k < T(n) \leq k+1;$$

În concluzie, numărul maxim de operații este: $k+1=\lceil \log_2 n \rceil + 1$

✓ Există și cazuri în care explorarea sistematică returnează rezultatul după mai puține comparații. De exemplu, dacă prima componentă conține valoarea căutată.

✓ Ar fi interesant să completați programul care efectuează căutarea binară ca să tipărească numărul de comparații efectuate.

Prin urmare, căutarea binară este un algoritm în **O(log(n))**.

7.2 Ce trebuie să mai știm...

⇒ *Există probleme pentru care nu se cunosc decât algoritmi de complexitate exponențială.* În această carte vom prezenta câțiva astfel de algoritmi.

Așa cum am arătat, algoritmii de complexitate exponențială nu sunt de mare folos, chiar dacă dispunem de supercalculatoare. Totuși, problema trebuie rezolvată într-un fel. De multe ori, astfel de probleme cer să se găsească o soluție optimă (adică o soluție care maximizează sau minimizează o anumită funcție). În astfel de cazuri, se aplică alți algoritmi, neexponențiali, care nu garantează găsirea unei soluții optime, dar găsesc o soluție apropiată de aceasta. Astfel de algoritmi se numesc *algoritmi euristici.*

⇒ Atunci când elaborăm un algoritm, căutăm ca acesta să aibă o complexitate cât mai mică, pentru a obține soluția cât mai repede. Astfel, preferăm un algoritm în **O(n)** unuia în **O(n^2)**, preferăm un algoritm în **O(log(n))** unuia în **O(n)**.

⇒ Chiar dacă doi algoritmi au aceeași complexitate, ei pot avea o eficiență diferită. Fie doi algoritmi echivalenți (la aceeași intrare au aceeași ieșire). Dacă unul efectuează **n** comparații are complexitatea **O(n)**, iar dacă celălalt efectuează **2×n** comparații are tot complexitatea **O(n)**, dar, evident, va fi preferat primul algoritm.

⇒ În cazul în care un algoritm efectuează, pe rând, anumite operații de complexități diferite, se consideră că algoritmul are complexitatea maximă.

Exemplu: algoritmul efectuează, pe rând, două operații: una de complexitate **O(n)**, alta de complexitate de **O(n^2)**. Algoritmul are complexitatea **O(n^2)**.

Probleme propuse

1. Se dă un vector cu `n` componente numere naturale. Care dintre algoritmii de mai jos, studiați în anul precedent, nu are complexitatea `O(n)`?

a) Algoritmul pentru calculul valorii minime din vector;
b) Algoritmul care testează apartenența unui număr natural între valorile reținute de vector;
c) Algoritmul pentru calculul valorii maxime din vector;
d) Oricare algoritm studiat care sortează crescător valorile reținute de vector.

2. Care este complexitatea secvenței de mai jos?

```
for i:=1 to n do
 for j:=n downto 3 do A[i,j]:=A[i,j]*2;
```

a) `0`, pentru că nu se efectuează nici o comparație;
b) $O(n)$;
c) $O(n^2)$;
d) $O(n\times(n-3))$.

3. Presupunem că dispunem de un algoritm de sortare a `n` numere naturale care are complexitatea $O(n\times\log(n))$. Există un astfel de algoritm, îl veți învăța în clasa a X-a. Care este complexitatea unui algoritm optim prin care se decide dacă `n` numere naturale sunt distincte?

a) $O(n\times\log(n))$;
b) $O(n^2)$;
c) $O(n\times\log(n)+n)$;
d) $O(n)$.

4. Se dau doi vectori `A` și `B`. Fiecare reține `n` numere naturale distincte. Care dintre algoritmii de mai jos, studiați în anul precedent, are o complexitate diferită de a celorlalți `3`?

a) Algoritmul care calculează reuniunea mulțimilor reținute de cei doi vectori;
b) Algoritmul care calculează intersecția mulțimilor reținute de cei doi vectori;
c) Algoritmul care calculează apartenența unui element la reuniunea mulțimilor reținute de cei doi vectori;
d) Algoritmul care calculează diferența mulțimilor reținute de cei doi vectori.

5. Care este complexitatea algoritmului de sortare prin interschimbare (metoda bulelor)?

a) $O(n^2)$; **b)** $O(n \times \log(n))$; **c)** $O(n)$; **d)** $O(n^3)$.

6. Să observăm că algoritmul de sortare prin metoda interschimbării poate fi îmbunătățit, dacă ținem seama de faptul că, după o parcurgere a vectorului, cel mai mare (mic) element ajunge pe ultima poziție. La a doua parcurgere, este necesar să se parcurgă vectorul până la componenta **n-1** ș.a.m.d. Scrieți programul care implementează algoritmul îmbunătățit. Obțineți astfel o îmbunătățire a complexității?

7. Care este complexitatea algoritmului de interclasare, dacă se interclasează valorile reținute de doi vectori cu **m** și **n** componente?

8. Se dau **n-1** numere naturale distincte din mulțimea **{1,2...n}**. Se cere să se găsească numărul din mulțime care lipsește. Complexitate cerută: **O(n)**.

Răspunsuri:

1. d) Observație: la punctul **b)** nu se poate aplica căutarea binară, pentru că vectorul nu este sortat. **2. c) 3. a)** Indicație: Se sortează, apoi, într-o singură parcurgere a numerelor, se determină dacă există sau nu elemente distincte. **4. c) 5. a) 6.** Deși algoritmul este mai performant, complexitatea rămâne aceeași... **7. O(n+m) 8.** Se calculează suma celor **n-1** numere și această valoare se scade din suma primelor **n** numere naturale: $\frac{n(n+1)}{2}$.

CAPITOLUL 8

Ce este informatica ?

8.1 Scurt istoric al calculatorului

În lume, în fiecare moment sunt folosite milioane de calculatoare în cele mai diverse scopuri. Există oare domenii în care el nu și-a făcut simtită prezența? Nu, în secolul XXI viața oricărui om este influențată de calculator.

Informatica este o știință care a apărut în strânsă legătură cu această invenție. Omul și-a creat un instrument pentru a-și perfecționa munca de prelucrare a informațiilor. Inițial, calculatorul a fost folosit în special pentru efectuarea calculelor numerice, de aici și numele său (*to compute - a calcula*). Cu timpul, datorită dezvoltării ca știință a informaticii, s-a reusit prelucrarea și a altor forme de informație, reprezentate sub formă de text, sunet, imagine, etc.

În antichitate, în Asia Mică, *abacul* a reprezentat prima formă de instrument de calcul. Comercianții foloseau acest dispozitiv pentru calcularea valorilor tranzacțiilor. Cu timpul, datorită apariției hârtiei, a uneltelor de scris folosirea abacului a fost abandonată.

Prima mașină de calcul a fost realizată în secolul XVII de Blaise Pascal. Mașina inventată de el putea însuma numere de maximum **8** cifre. Era formată din 8 roți, fiecare având înscrise cifrele de la **0** la **9**. De fiecare dată când cifra unităților făcea o rotație completă, aceasta misca roata zecilor, ș.a.m.d.

Matematicianul *Charles Babbage*, considerat astăzi "părintele calculatorului", a proiectat în anul 1832 primul calculator multifuncțional programabil numit "Analythical Engine" (Motorul Analitic). Calculatorul urma să fie compus din peste 50000 de părți, era gândit ca fiind înzestrat cu memorie, putea fi programat și ar fi utilizat cartele perforate. Putea să rețină maximum **1000** de numere a câte **50** de cifre fiecare. Datorită lipsei de fonduri acest calculator n-a fost realizat din punct de vedere fizic. Babbage a colaborat cu *Augusta Ada Byron,* fiica cunoscutului poet. Ea a sugerat utilizarea sistemului binar și a buclelor (instrucțiuni care se repetă), motiv pentru care este considerată primul programator din lume. În 1854 *George Boole* publică o lucrare despre logica booleană (adevărat, fals). Ideile care care se găsesc în această lucrare stau la baza circuitelor care alcătuiesc calculatoarele actuale.

Primul calculator modern a fost realizat în 1946 în cadrul Universității din Pennsylvania. A fost numit **ENIAC** (Electronic Numerical Integrator and Computer). Avea în componența sa 18000 de lămpi, 70000 rezistențe și nu mai puțin de 5 milioane de lipituri cu cositor. Calculatorul era masiv și consuma în aproximativ 160 KiloWați.

John Von Neumann s-a alăturat în 1945 echipei din Pensylvania inițiind concepte arhitecturale care aveau să rămână valabile până în prezent. El este părintele primului model arhitectural pentru calculatoare moderne (**EDVAC**). Calculatorul dispunea de o memorie folosită pentru stocarea atât a programelor și a informațiilor.

Elementul cheie în arhitectura Von Neumann a fost Unitatea Centrală de Procesare (CPU) care permitea ca toate funcțiile calculatorului să fie coordonate dintr-o singură sursă.

Acesta a fost practic momentul când calculatorul și informatica aveau să intre definitiv în istoria omenirii.

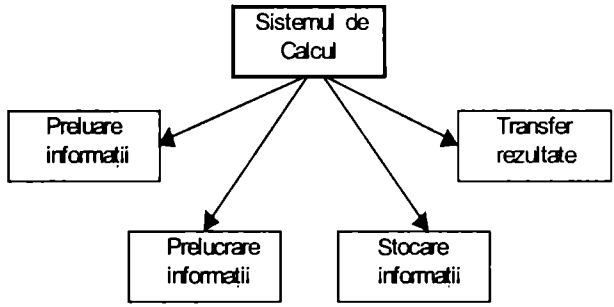
8.2 Ce este informatica ?

Informatica este știința care se ocupă cu prelucrarea informațiilor folosind sistemele automate de calcul.

Ea s-a dezvoltat ca știință începând cu anii '50 în mod deosebit în cadrul universităților, având ca scop inițial studiul și perfecționarea calculatoarelor. Cu timpul s-a dezvoltat și creat o deosebită literatură de specialitate, apărând anumite departamente de cercetare care au generat mai apoi domeniile informaticii de azi. Dintre acestea amintim:

- Arhitectura calculatoarelor;
- Sisteme de operare;
- Limbaje de programare;
- Algoritmi și structuri de date ;
- Sisteme de gestiune a bazelor de date.

Un sistem de calcul este ansamblul format din componente fizice(hardware) și logice(software). Funcțiile sistemului de calcul derivă din funcțiile pe care acesta trebuie să le asigure:



8.3 Rolul informaticii în dezvoltarea societății

Calculatorul și informatica a produs schimbări majore în evoluția omenirii. Datorită dezvoltării informaticii și a ingineriei programării, calculatorul a reușit să preia nu numai munca de rutină a omului, ci să conducă la schimbări majore și ireversibile în viața lui.

Este indispensabil în sectoarele economice, administrațiile financiare, în controlarea și modelarea proceselor tehnologice, în medicină, în educație, în domeniile de cercetare, etc.

O dată cu dezvoltarea informaticii s-au creat meserii noi: analiști, programatori, operatori, ingineri de sistem. De altfel, utilizarea calculatorului a devenit indispensabilă și în cadrul altor calificări profesionale (ziariști, contabili, cercetători, medici, ingineri, etc).

A înlocuit prezența fizică a omului în activități periculoase, acestea fiind controlate de calculator (diverse misiuni spațiale, testarea unor prototipuri de vehicule, activități în centrale nucleare, etc).

A îmbogățit vocabularul universal prin cuvinte noi (hardware, software, scanner, etc) sau sensuri noi ale cuvintelor (memorie, director, program, etc).

CAPITOLUL 9

Recapitularea prin teste grilă a cunoștințelor însușite în clasa a IX-a

1. Nu întotdeauna în pseudocod se declară tipul variabilelor. Care este tipul variabilei **m** pentru programul C++ rezultat din pseudocodul de mai jos:

```
citește m număr natural;
m←m+13.13;
scrie m
```

a) int;
b) float;
c) pseudocodul este greșit, pentru că **m** este citit ca număr natural și este utilizat ca real;
d) pseudocodul este ambiguu.

2. Dacă **a**, **b**, **c** sunt variabile întregi, ce va afișa secvența de mai jos:

```
a←3; b←a+2; c←b-3; a←a+b-c; scrie a, b, c
```

a) 3 5 2; **b)** 5 3 4; **c)** 6 0 3; **d)** 6 5 2.

3. Variabilele **a, b, c** sunt de tip întreg și reține, în această ordine, valorile **3**, **4**, **1**. În ce ordine trebuie efectuate atribuirile de mai jos, astfel încât, la sfârșit, **a** să rețină **-1**, **b** să fie egal cu **a**, iar **c** să rețină o valoare egală cu **b+1**.

i) c←b+c; ii) a←b-c; iii) b←a-b;

a) iii) i) ii)
b) ii) i) iii)
c) ii) iii) i)
d) i) ii) iii)

4. În condițiile în care **a** reține **3**, **b** reține **4**, **c** reține **5** și se execută secvența următoare, care din atribuirile de mai jos trebuie eliminată (una singură) astfel încât **a** să rețină **3**, **b** să rețină **-1** și **c** să rețină **12** ?

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

a) c←a+b+c-1; **b)** a←a+b-c+1;
c) c←a+b+c; **d)** b←a-b.

5. Care dintre secvențele de mai jos interschimbă conținutul a două variabile numerice **a** și **b**? Se știe că **m** este variabilă întreagă și **^** este operația **sau exclusiv**.

a) **a←b; b←a;**
b) **a←b; m←a; b←m;**
c) **m←a^b; a=a^m;b=b^m;**
d) **a←a+b; b←a-b; a←b-a.**

6. Care dintre atribuirile de mai jos nu este echivalentă cu **a←b**? Se știe că **a** și **b** sunt variabile de tip întreg, **^**, **|** și **&** au semnificația operatorilor cu același nume din C++.

a) **a←a^a^b;**
b) **a←a^(a^b);**
c) **a← a&b + a|b;**
d) **a←a+(b-a).**

7. Ce afișează programul următor?

```
#include <iostream.h>
main()
{ int a; float b;
  b=7;
  a=b;
 cout<<a;
}
```

a) 7.0;
b) programul nu funcționează, va da eroare de sintaxă;
c) 7;
d) 0, pentru că operația de atribuire nu se poate executa.

8. Ce afișează programul următor?

```
#include <iostream.h>
main()
{ int a; float b,c;
  b=7;c=2;
  a=b/c;
  cout<<a;
}
```

a) programul nu funcționează, va da eroare de sintaxă;
b) 3.5;
c) 3;
d) 3.0

Testele de la **9** la **14** se referă la pseudocodul următor, unde **n** este număr natural din mulțimea **[1,10]**. Prin **|a|** înțelegem modulul unui număr **a**, iar **a % b** înțelegem restul împărțirii întregi a lui **a** la **b**, calculat la fel ca în C++.

```
citește n
pentru i=1,n execută
|  citește V[i] (număr întreg)
|_■
s←|v[1] % 10|;
m←v[1];
pentru i=2,n execută
| dacă Vi>m atunci
| | m←V[i];
| | s←s+|v[i] mod 10|;
| |_■
|_■
scrie s
```

9. Dacă se citesc, în această ordine, valorile **4**, **1**, **2**, **3**, **4** atunci se va afișa:

a) 10 **b) 14** **c) 0** **d) 4**

10. Ce se afișează dacă se citesc, în ordine, valorile: **5, 3, 1, 12, 4, 25**?

a) 15 **b) 17** **c) 45** **d) 10**

11. Ce se afișează dacă se citesc, în ordine, valorile: **4, -3, -2, -5, -1**?

a) -6 **b) 6** **c) eroare** **d) 11**

12. Ce se afișează dacă se citesc, în ordine, valorile: **3, 2, 5?**

a) 7;
b) 10;
c) eroare de sintaxă;
d) se așteaptă citirea unei alte valori.

13. Care dintre valorile de mai jos trebuie citită pentru ca să se afișeze **7** în cazul de mai jos: **4 12 1 ? 15**? Valoarea citită trebuie să înlocuiască **"?"**.

a) 7 **b) 13** **c) 14**
d) nu există o valoare care îndeplinește cerința.

14. Care dintre variantele de mai jos este un program **C++** echivalent cu pseudocodul inițial? Notă: *două programe sunt echivalente dacă la aceleași intrări au aceleași ieșiri.*

a)

```
#include <iostream.h>
#include <math.h>
main()
{ int n,i,s,m,a;
  cout<<"n=";cin>>n;
  cin>>a;
  s=abs(a%10); m=a;
  i=2;
  while(i<=n)
  { cin>>a;
    if (a>m)
    { m=a;
     s+=abs(a%10);
    }
  i++;
  }
  cout<<s;
}
```

b)

```
#include <iostream.h>
#include <math.h>
main()
{  int n,i,s,m,a;
   cout<<"n=";cin>>n;
   cin>>a;
   s=abs(a%10); m=a;
   i=2;
  do
  { cin>>a;
    if (a>m)
    { m=a;
      s+=abs(a%10);}
  i++;
  } while (i<=n);
  cout<<s;
}
```

c)

```
#include <iostream.h>
main()
{ int V[11],n,i,s,m;
  cout<<"n=";cin>>n;
  for     (i=1;i<=n;i++)
cin>>V[i];
  s=V[1]%10;m=V[1];
  for(i=2;i<=n;i++)
    if (V[i]>m)
     { m=V[i];
      s+=V[i]%10; }
  cout<<s;
}
```

d)

```
#include <iostream.h>
#include <math.h>
main()
{ int V[11],n,i,s,m;
  cout<<"n=";cin>>n;
  for    (i=1;i<=n;i++)
cin>>V[i];

s=abs(V[1]%10);m=V[1];
  for(i=2;i<=n;i++)
    if (V[i]>m)
     { m=V[i];
     s=abs(s+V[i]%10);}
  cout<<s;
}
```

Itemii de la **15** la **18** se referă la pesudocodul următor, unde prin **mod** și **div** înțelegem operatorii prin care se obțin restul și, respectiv, câtul împărțirii a două numere întregi:

```
citește n (număr natural<100000)
repetă
| s←0;
| cât timp n ≠ 0
| | s←s+n mod 10;
| | n←n div 10
| |_■
| n←s
|_■ până când n<10
scrie s
```

15. Care este declarația din C++ care utilizează minimum de memorie pentru a declara variabilele **n** și **s**?

a) `int n,s;`
b) `long n; char s;`
c) `int n; char s;`
d) `unsigned int n; unsigned int s;`

16. Care este valoarea care se afișează dacă pentru **n** se citește **88888**?

a) 40 **b) 8** **c) 4** **d) 5**

17. Pentru care valoare citită, dintre cele de mai jos, se afișează **7**?

a) 4372 **b) 8765** **c) 9993** **d) 458**

18) Spunem că două valori de intrare ale lui **n** sunt *din aceeași clasă* dacă pentru fiecare dintre ele algoritmul returnează aceeași valoare. Care din următoarele perechi de numere sunt din aceeași clasă?

a) 789 345 **b) 1234 811** **c) 9 64** **d) 88 99**

19. Care dintre expresiile de mai jos este echivalentă cu expresia:

`!(a>b && c>d)`

Variabilele **a, b, c, d** sunt de tip întreg.

a) `a<b&&c<d` **b)** `a<=b&&c<=d` **c)** `a<=b||c<=d` **d)** `a>=b)||c>=d`

20. Variabilele **a, b, c, d** sunt de tip întreg. Care din structurile de mai jos (scrise în C++) este echivalentă cu structura următoare:

```
if (a>b & a>c)
 cout<<"OK";
 else cout<<"NO";
```

a)

```
if (a>b)
   if (a>c)
    cout<<"OK";
    else cout<<"NO";
   else cout<<"NO";
```

b)

```
if (a>b)
   if (a>c)
    cout<<"OK";
    else cout<<"NO";
```

c)

```
if (a>b)
   if (a>c)
     cout<<"OK"
    else cout<<"NO"
   else cout<<"NO";
```

d)

```
if (a>b)
  if (a>c)
    cout<<"OK";
    else
   else cout<<"NO";
```

21. Care afirmație din cele de mai jos, toate referitoare la algorimul următor, este falsă? Observație: operatorul % are aceeași semnificație cu operatorul % din C++:

```
citește n
pentru i=1,n execută
|  citește V[i] (număr natural)
|_■
x←v[1] % 10;
pentru i=2,n execută
| x←(x*(v[i]%10)) mod 10;
|_■
scrie x
```

a) Calculează restul împărțirii la 10 a produsului elementelor unui vector citit;
b) Calculează ultima cifră (cifra cea mai puțin semnificativă) a produsului elementelor vectorului citit;
c) Calculează produsul numerelor formate din ultimă cifră a fiecărei valori reținută de o componentă;
d) Calculează cea mai mică valoare care, dacă se scade din produsul elementelor vectorului citit se obține un număr divizibil cu 10.

22. Priviți secvențele de mai jos. Variabila n este de tip int, iar n este număr natural. Dintre ele 3 și numai 3 sunt echivalente. Care este secvența care nu este echivalentă cu celelalte?

```
a) if (n%2==0) cout<<"OK";
b) if (n/2.==n/2) cout<<"OK";
c) if (n>>1<<1==n) cout<<"OK";
d) if (n<<1>>1==n) cout<<"OK".
```

23. Se citește **n** număr natural, **n≥1**. Care dintre secvențele de mai jos, ciclează? Observație: a cicla înseamnă a rula la infinit.

a)

```
citeşte n
s←0; i←1;
cât timp i ≤ n
 | s←s+i;
 | n←n-1
 |_■
scrie s
```

b)

```
citeşte n
s←0; i←1;
cât timp i ≤ n
 | s←s+i;
 | i←i+1
 |_■
scrie s
```

c)

```
citeşte n
s←0; i←1;
cât timp i ≤ n
 | s←s+i;
 | n←i
 |_■
 scrie s
```

d)

```
citeşte n
s←0; i←1;
cât timp i ≤ n
 | s←s+i;
 | n←i-1
 |_■
scrie s
```

24. Se citește un număr natural **n>0**. De câte ori se execută instrucțiunile incluse în structura **Repetă ... cât timp**, dacă se citește **n=345** ? Prin **a%b** se înțelege împărțirea întreagă a lui **a** la **b**.

```
citeşte n
s←0; i←1;
Repetă
 | s←s+1;
 | n←n%10;
 |_■ cât timp (s<=2) şi (n≠0)
 scrie s
```

a) 0 **b) 1** **c) 2** **d) 3**

25. Ce afișează secvența următoare, dacă se citește o valoare reală pozitivă? Prin $[n]$ am notat partea întreagă a numărului **n**.

```
citeşte n
cât timp n ≠ 0
 | n←n-[n]
 | n←n*10
 | scrie [n]
 |_■
```

a) toate zecimalele numărului n, așa cum s-a introdus numărul;
b) zecimalele numărului n, așa cum a fost memorat;
c) zecimalele numărului n, dar programul poate cicla dacă n are o infinitate de zecimale;
d) se afișează, pe rând, cifrele care alcătuiesc $[n]$.

26. Pentru a calcula maximul (minimul) dintre n valori neordonate sunt necesare:

a) exact n-1 comparații;
b) cel mult n-1 comparații;
c) cel puțin n-1 comparații;
d) depinde de metoda de sortare utilizată.

27. Avem două fișiere și fiecare dintre ele conține, în ordine alfabetică, numele elevilor unei clase. Se cere să obținem un singur fișier cu numele elevilor în ordine alfabetică. Care variantă o alegeți pentru a lucra eficient?

a) se interclasează valorile din cele două fișiere;
b) se concatenează valorile și se alege un algoritm de sortare convenabil;
c) se sortează datele din fiecare fișier și apoi se interclasează valorile din cele două fișiere;
d) se interclasează fiecare fișier și apoi valorile astfel rezultate se concatenează.

28. Se consideră un vector care reține n valori reale ordonate crescător. Se cere să elaborați un algoritm care să decidă dacă cele n valori sunt distincte. Ce alegeți?

a) se parcurge o singură dată vectorul și se efectuează n-1 comparații;
b) se parcurge de mai multe ori vectorul și se efectuează $\frac{n(n-1)}{2}$ comparații;
c) dacă sunt sortate, valorile sunt distincte;
d) Problema este nedecidabilă.

29. Un algoritim care testează apartenența unui număr real la o mulțime formată din n valori reale, neordonate, valori reținute într-un vector, efectuează:

a) exact n-1 comparări;
b) cel mult n-1 comparări;
c) exact n comparări;
d) cel mult n comparări.

30. Se consideră un vector care conține **n** valori, numere întregi, într-o ordine oarecare. Un algoritm testează dacă aceste valori alcătuiesc sau nu o mulțime (sunt sau nu distincte). Care este numărul minim de comparări care vor fi efectuate:

a) **n-1** comparări;
b) **0** comparări;
c) $\frac{n(n-1)}{2}$ comparări;
d) orice vector cu **n** componente (evident, de acelasi tip) retine o multime cu **n** elemente, deci nu este necesar un algoritm pentru aceasta.

Răspunsuri:

1. b) Nu trebuie confundat tipul unei variabile cu valoarea inițială pe care o reține. Trebuie găsit un tip care este capabil să rețină toate valorile care pot interveni pe parcursul executării programului.
2. d) 3. a) 4. a) 5. c) 6. c) 7. c) 8. c) 9. a) 10. d)
11. b) 12. d) 13. a) 14. a)

✓ Foarte important! Un grup de astfel de teste, așa cum a fost acesta, poate fi rezolvat cu mult mai ușor dacă realizați ce face algoritmul respectiv. În exemplu, acesta calculează suma ultimelor cifre pentru subșirul strict crescător care începe cu primul element al vectorului. De exemplu, dacă **n=4** și se citesc, pe rând, valorile **-5, 3, 1, 7** se va afișa **5+3+7=15** (elementul **1** nu face parte din subșir). Dacă observați aceasta, toți itemii se rezolvă imediat.

15. b) 16. c) 17. a) 18. b)

Algoritmul calculează suma cifrelor unui număr, apoi din nou suma cifrelor lui până când suma obținută este mai mică decât **10**. Exemplu: **n=678; 6+7+8=21 2+1=3,** deci valoarea afișată este 3. Cât de ușor se rezolvă itemii de la **15** la **18** dacă observați aceasta....

19. c) Avem: **!(a>b && c>d) = !(a>b) || !(c>d)= a<=b || c<=d;**

20. a) 21. c) 22. d) 23. c) 24. d) 25. b)

26. a) 27. a) 28. a) 29. d) 30. c) (trebuie testat dacă elementele sunt distincte).

ANEXA 1

Mediul limbajului de programare studiat

1.1. Prezentare generală

Pentru a ușura munca programatorilor, de regulă, firmele livrează limbajele sub forma unui mediu integrat de programare. Acesta este alcătuit în principal din:

- editor de texte;
- compilator;
- o interfață cu meniuri.

și are rolul de a permite scrierea cu ușurință a programelor.

Editorul de texte este acea componentă software care permite introducerea textelor diverselor programe.

Compilatorul este acea componentă software care traduce textul programului introdus de noi în limbaj mașină (succesiune de 0 și 1).

Prin meniu se înțelege o succesiune de cuvinte, fiecare cuvânt având semnificația de comandă dată calculatorului, pentru ca acesta să efectueze o anumită operație.

Numele provine de la faptul că, pe ecran, apar toate cuvintele, utilizatorul putând să selecteze acel cuvânt care corespunde comenzii pe care urmează să o dea calculatorului. Un cuvânt din meniu (cuvântul este în limba engleză) simbolizează operația pe care o va efectua calculatorul dacă acesta a fost selectat. Selecția se efectuează prin plimbarea unei bare luminoase asupra cuvintelor din meniu, urmată de apăsarea tastei Enter.

1.2. Editarea programelor sursă

1.2.1. Utilizarea meniului

În partea de sus a ferestrei apare meniul cu care este înzestrat mediul integrat al limbajului. Astfel apar opțiunile următoare:

FILE EDIT SEARCH RUN COMPILE DEBUG OPTIONS WINDOW HELP

Avem două posibilități de a selecta o opțiune:

- se tastează **F10**, apoi ne deplasăm cu ajutorul săgeților pe una din opțiunile dorite;
- se tastează **ALT+litera** de început a opțiunii (de exemplu, pentru **FILE** tastăm **ALT+F**);

În ambele cazuri, după poziționarea pe o anumită opțiune se tastează **ENTER** iar pentru revenire se tastează **ESC**.

1.2.2. Salvarea și încărcarea programelor

În paragraful anterior am învățat să scriem (editățm) un program. Textul sursă al programului, se găsește în memoria internă. Imediat ce părăsim mediul integrat (ALT+x), acesta se pierde.

Să presupunem că după o zi, două, dorim să rulăm din nou programul anterior. Ce avem de făcut? Introducem iar textul sursă și efectuăm operațiile care au fost arătate? Se poate și așa, dar pierdem timp.

Pentru ca aceasta să nu se întâmple, odată introdus, textul sursă al unui program trebuie să fie *salvat*.

Prin salvarea unui program se înțelege operația prin care acesta este scris pe suport extern (de cele mai multe ori pe hard-disc, dar și pe dischetă) *sub formă de fișier*.

Pentru a putea refolosi programul salvat, acesta trebuie încărcat.

Prin încărcarea unui program se înțelege operația prin care este citit de pe suportul magnetic și introdus în memoria internă (acesta devine vizibil pe ecran).

Salvarea unui program se face utilizând un anumit *nume al programului* (eventual și calea). De asemenea, încărcarea sa se face după *numele* cu care a fost salvat.

Numele sub care se face salvarea sau încărcarea programelor este numele ferestrei care îl conține, urmat de extensie.

Până în acest moment nu am folosit decât nume implicit (dat de sistem) pentru fereastra program. Este bine ca salvarea programului să se facă sub nume alese de noi (explicite). Să presupunem că am introdus textul sursă al unui program într-o fereastră cu nume implicit. Pentru a salva programul, vom proceda astfel:

- apelăm opțiunea **FILE** a meniului (**ALT+F**);
- se selectează opțiunea **Save As** (se poziționează bara pe opțiunea arătată, după care se tastează **Enter**);
- apare o fereastră cu un rol special (pentru salvare) - în care tastăm numele sub care vrem să salvăm programul.

În urma acestor acțiuni, programul se va găsi pe hard-disc sau dischetă.

Este posibil ca să dorim să salvăm un program aflat într-o fereastră cu nume explicit și să nu dorim schimbarea numelui. Acest lucru este posibil. Vom proceda astfel:

- apelăm opțiunea **FILE** a meniului (**ALT+f**);
- selectăm opțiunea **Save**.

Mai simplu, salvarea se face apăsând tasta **F2**.

Diferența între **Save** și **Save As** constă în faptul că **Save** *salvează un program cu numele pe care îl are fereastra*, iar **Save As** *salvează programul cu un nou nume (care va fi introdus de programator)*. Totuși, dacă fereastra are un nume implicit, programatorul este întrebat cu ce nume să se facă salvarea. De asemenea, în cazul în care se dorește să se salveze un program cu un nume cu care mai este salvat un altul, programatorul este avertizat de acesta și întrebat dacă salvează peste cel existent.

Observație. Este bine ca atunci când scriem un program să-l salvăm din când în când (poate fi o întrerupere de curent sau alt incident, cazuri în care am muncit degeaba).

Observație. Fără altă opțiune, salvarea unui program se face în directorul curent. În cazul în care se dorește salvarea în alt director, trebuie precizată și calea (pentru **Save As**).

Să presupunem că dorim să încărcăm programul salvat anterior. Se poate proceda astfel:

- se selectează opțiunea **FILE** a meniului (**ALT+f**);
- se selectează opțiunea **OPEN**;
- se tastează numele programului, iar dacă este cazul, acesta este precedat de cale.

  Mai simplu, se tastează **F3**, apoi numele.

De fapt, indiferent de calea pe care o alegem, apare o așa numită *fereastră de dialog* în care introducem numele specificat. După apariția ferestrei de dialog, putem proceda și altfel. Vom observa că, în partea inferioară a ei, se găsește o listă care cuprinde:

⇒ lista programelor din folder-ul curent;

⇒ lista subdirectoarelor folder-ului curent (se recunosc prin faptul că se termină cu caracterul '****' (exemplu: **BGI**);

⇒ caracterele **..** (au semnificația de folder părinte).

Saltul la această listă se poate face tastând **TAB**. Odată efectuat saltul, putem explora întreg hard-ul, pentru a încărca programul pe care îl dorim. Cum procedăm?

- Dacă programul pe care dorim să-l încărcăm se găsește în directorul curent, plimbăm cu ajutorul tastelor săgeți o bară luminoasă până aceasta se poziționează pe numele programului, apoi tastăm **Enter**.
- Dacă programul se găsește într-un subdirector al directorului rădăcină, poziționăm bara luminoasă asupra numelui său, apoi tastăm **Enter** și procedăm ca în cazul anterior.
- Dacă programul se găsește în directorul părinte sau într-un subdirector al acestuia, explorăm directorul părinte plasând bara luminoasă

deasupra caracterelor **..** și tastând **Enter**, apoi procedăm ca în cazurile anterioare.

Astfel se evită tastarea numelui complet (unitatea, calea și nume).

1.2.3. Lucrul cu mai multe ferestre program

Există momente când *este necesar să avem, simultan, mai multe programe în memorie* (unele le încărcăm de pe hard-disc, altele le scriem în acel moment etc.).

Mediul integrat de programare permite acest lucru. De exemplu, putem încărca (așa cum a fost arătat în paragraful anterior) două programe de pe hard-disc. Evident, fiecare program are asociată fereastra sa (numele ferestrei coincide cu numele programului care a fost încărcat).

În cazul existenței mai multor ferestre program, *la un anumit moment, numai una este activă* (*în sensul că putem opera numai asupra programului aflat în fereastra activă*).

Să presupunem că avem 3 ferestre program la un moment dat în memorie (am arătat faptul că numai una din ele este activă). *Pentru ca altă fereastră să devină activă, se selectează opțiunea* **NEXT** *a meniului* **Window**. *Mai simplu, se tastează* **F6**. Repetând procedeul următoarea fereastră devine activă, până când prima fereastră devine din nou activă. O modalitate mai rapidă de a activa o fereastră este de a utiliza combinația de taste **ALT +cifra**, unde **cifra** indică numărul ferestrei, prezent în colțul stânga sus.

Există cazuri când dorim să introducem într-un program o secvență, dintr-un alt program, deja introdus. Pentru a putea efectua această operație, *trebuie să marcăm ca bloc secvența respectivă* (odată marcată, va figura în program într-un context grafic diferit). Pentru marcare, procedăm astfel:

⇒ se poziționează cursorul la începutul primului rând al secvenței;

⇒ apăsăm tasta Shift în mod continuu și cu săgeata jos (↓) marcăm rândurile care o alcătuiesc.

Odată marcat, un bloc poate fi transferat într-o fereastră specială numită **CLIPBOARD**. Transferul se poate realiza în două moduri:

- blocul este copiat în **CLIPBOARD** (rămâne și în fereastra care îl conține);
- blocul este mutat în **CLIPBOARD** (dispare din fereastra care îl conține).

În primul caz, se apelează opțiunea **COPY** a meniului **Edit** (**Alt+e** și se selectează **COPY**) sau, mai scurt **Ctrl+Ins**.

În al doilea caz se apelează opțiunea **CUT** a meniului **EDIT** (**ALT+E** și se selectează **CUT**) sau **SHIFT+DEL**.

Dacă dorim, putem vizualiza conținutul ferestrei **CLIPBOARD** apelând opțiunea **SHOW CLIPBOARD** a meniului **EDIT**.

Din **CLIPBOARD**, un bloc poate fi inserat în fereastra activă. Inserarea se face, *începând cu poziția curentă a cursorului*, apelând opțiunea **PASTE** a meniului **EDIT** (pe scurt, **Shift+Ins**).

Putem modifica dimensiunile ferestrei active. Aceasta se face astfel:

⇒ se apelează opțiunea **SIZE MOVE** a meniului **WINDOW** (marginile ferestrei active capătă altă culoare);

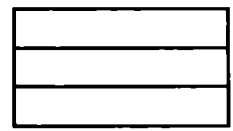
⇒ cu ajutorul tastei **Shift** + săgețile dreapta (→), stânga (←) se deplasează latura din dreapta a dreptunghiului care marchează fereastra (cea din stânga rămâne fixă);

⇒ cu ajutorul tastei **Shift** + săgețile sus (↑), jos (↓) se deplasează baza dreptunghiului (latura de sus rămâne fixă);

⇒ după ce am adus fereastra la dimensiunile dorite, ea poate fi mutată cu ajutorul săgeților în poziția dorită;

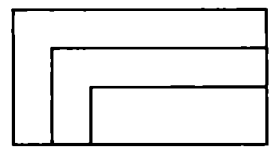
⇒ în final, se tastează **Enter**.

În cazul în care dorim, putem modifica aranjarea ferestrelor program pe ecran.



Le putem aranja una sub alta cu ajutorul opțiunii **TILE** a meniului **WINDOW**.

Le putem aranja în cascadă cu ajutorul opțiunii **CASCADE** a meniului **WINDOW**.



Dacă dorim ca fereastra activă să ocupe întreg ecranul se apelează opțiunea **ZOOM** a meniului **WINDOW**.

1.2.4. Alte facilități de editare

Opțiunea **SEARCH\FIND**

Are rolul de a căuta un șir în textul programului sursă. În mod practic, se afișează o fereastră în care suntem solicitați să introducem textul care se va căuta. De asemenea, în fereastră se prezintă alte câteva opțiuni (se ajunge la ele prin apăsarea tastei **TAB** și se marchează cu spațiu):

- **CASE SENTIVE** – dacă este marcată (cu **X**) se face distincție între literele mari și mici;
- **WHOLE WORDS ONLY** – consideră text ca un cuvânt separat (prin blancuri sau alți separatori);
- **REGULAR EXPRESION** – textul poate conține și caractere de control;
- **DIRECTION** – stabilește direcția în care se face căutarea:

- ◊ **FORWARD** – spre sfârșit;
- ◊ **BACKWARD** – spre început;

- **ORIGIN** - stabilește locul de unde începe căutarea:
  - ◊ **FROM CURSOR** – începând de la cursor;
  - ◊ **ENTIRE SCOPE** – întregul domeniu.
- **OK** – s-au terminat opțiunile, se închide fereastra și se face căutarea;
- **CANCEL** – se renunță la căutare;
- **HELP** - se dau informații despre căutare.

Opțiunea **SEARCH\REPLACE**

Înlocuiește un șir cu altul. În acest sens, se deschide o fereastră care permite să se scrie șirul care se înlocuiește și șirul cu care se va înlocui. În această fereastră se găsesc opțiunile care au fost prezentate anterior și altele care urmează:

- **PROMPT ON REPLACE** – la fiecare înlocuire se deschide o fereastră în care utilizatorul este întrebat dacă să se facă sau nu înlocuirea;
- **OK** – se face o singură înlocuire;
- **CHANGE ALL** – se înlocuiesc toate aparițiile subșirului căutat.

Opțiunea **SEARCH AGAIN** – permite reluarea ultimei căutări.

Opțiunea **GO TO LINE NUMBER** – permite deplasarea cursorului pe o anumită linie (linia și coloana în care este cursorul sunt permanent afișate).

1.3. Compilare, rulare, depanare

Pentru a rula un program se tastează **CTRL+F9**. Dacă nu am tastat corect textul, apare un mesaj de eroare, iar cursorul se poziționează în locul unde se presupune că există eroarea (nu întotdeauna locul presupus este cel în care s-a produs într-adevăr). În acest caz, se efectuează corecția necesară și se încearcă, din nou, rularea programului (**CTRL+F9**). În cazul în care nu avem erori, se execută programul.

Procedând astfel, calculatorul execută mai multe acțiuni (corect, acestea se numesc *faze*), pe care le prezentăm în continuare.

- Compilarea programului - traducerea lui din program sursă în cod mașină. Dacă programul este tastat corect, compilarea reușește și se trece la faza următoare, altfel apare mesaj de eroare, așa cum am arătat (se numește eroare de sintaxă).
- Editarea legăturilor - fază pe care nu o prezentăm.

- Executarea programului - fază specifică fiecărui program în parte și pe care o prezentăm, în continuare, pentru programul de mai sus.

Să observăm că, în momentul executării, de pe ecran dispare textul programului, citirea și scrierea datelor se efectuează pe alt fond de ecran, iar după terminarea executării, pe ecran apare, din nou, textul sursă. În acest fel vom lucra cu două ferestre ecran:

- *fereastra în care găsim textul sursă (cea de mai sus);*
- *fereastra în care programul citește datele și scrie rezultatele* (**User Screen**).

Așa cum am văzut, după terminarea executării, se revine la imaginea în care se găsește textul sursă al programului. De cele mai multe ori, nici nu avem timpul necesar să vedem rezultatul tipărit. Pentru a-l vizualiza, se tastează **ALT+F5**. În acest caz, revenirea în fereastra în care se găsește textul sursă se face prin apăsarea tastei **Enter** sau, din nou, **ALT+F5**.

Atunci când încercăm să rulăm un program pot apărea o serie de erori. O parte sunt depistate în faza de compilare -erori de sintaxă-, o alta la rularea propriu-zisă.

Cum depistăm erorile care apar la executare?

- Pentru aceasta apăsăm tasta **F8** (se poate și cu **F7** deși există anumite diferențe între ele, dar nu în ce ne interesează în acest moment). Deasupra primei linii apare o bară luminoasă.
- Apăsăm din nou F8. Bara luminoasă se deplasează poziționându-se asupra următoarei linii.
- . . .

O astfel de rulare, în care calculatorul nu trece la executarea comenzilor de pe o linie decât după ce tastăm **F7** *se numește rulare pas cu pas.*

Rularea pas cu pas este extrem de utilă în *depanarea programelor* (corectarea erorilor care apar la executarea), dar și atunci când învățăm să programăm.

Rulând pas cu pas un program avem posibilitatea să verificăm modul în care se execută instrucțiunile. Se recomandă, la început, programele să fie rulate pas cu pas.

Observație: bara luminoasă se poziționează asupra unei linii și indică faptul că urmează să se execute instrucțiunile aflate pe acea linie (am văzut faptul că o linie poate conține mai multe instrucțiuni). Din acest motiv, pentru a *vizualiza executarea fiecărei instrucțiuni, este bine ca instrucțiunile să fie plasate câte una pe linie.*

Încă un avantaj pe care îl avem dacă rulăm pas cu pas un program: *putem vizualiza conținutul variabilelor chiar pe parcursul executării*

programelor. Aceasta ne ajută mult în învățarea programării. Atunci când dorim să vizualizăm o variabilă procedăm astfel:

⇒ tastăm **ALT+D**;
⇒ selectăm **EVALUATE/MODIFY**;
⇒ se tastează numele variabilei al cărei conținut dorim să-l vizualizăm.

De multe ori suntem siguri că o primă secvență de program funcționează corect (de exemplu o secvență de introducere a datelor). Cu toate acestea, programul funcționează incorect. Ce avem de făcut? Dacă rulăm pas cu pas programul, până la secvența pe care o dorim să o analizăm, pierdem timpul și răbdarea ne este pusă la grea încercare. Din acest motiv, este bine să trecem repede peste secvența de care suntem siguri.

Pentru a opri rularea programului acolo unde începe secvența pe care dorim să o testăm, trebuie să creăm un punct de întrerupere.

Cum procedăm? Plasăm cursorul pe linia din textul sursă la care dorim să se întrerupă executarea programului. Lansăm spre executare programul cu ajutorul tastei **F4**. Dacă suntem interesați, după aceasta putem muta din nou cursorul pe o nouă linie la care, când se ajunge, dorim să ne oprim.

Tehnica o putem utiliza combinat cu afișarea conținutului variabilelor. De asemenea, dintr–un punct de întrerupere se poate porni rularea pas cu pas a programului (cu tastele **F7** sau **F8**, după cum se dorește intrarea în subprograme sau nu). De fapt, cele prezentate până acum constituie opțiuni ale meniului **RUN**.

Opțiunea **PROGRAM RESET.** Considerăm un program care ciclează - anumite instrucțiuni se execută la infinit. De multe ori, din neatenție, facem astfel de greșeli. Trebuie să găsim o modalitate să–l oprim. Pentru aceasta se tastează **CTRL+BREAK(PAUSE)**. Programul se întrerupe, iar pe una din liniile textului sursă apare o bară care indică instrucțiunea la care s–a oprit programul nostru. Această informație ne este de mare folos, deoarece așa putem identifica secvența unde programul ciclează. Dacă dorim să relansăm programul în executare (eventual cu alte date de intrare, ca să vedem dacă și în acest caz programul ciclează), constatăm că rularea programului pornește din punctul unde aceasta a fost întreruptă (lansarea a fost făcută cu **CTRL+F9**). Deci tentativa noastră eșuează. Avem însă posibilitatea de a relua executarea, dacă selectăm, în prealabil, opțiunea **PROGRAM RESET**.

Crearea unui punct de întrerupere se poate face și cu ajutorul opțiunii **GO TO CURSOR** (în loc de **F4**).

Există și posibilitatea creării mai multor puncte de întrerupere, de la început. Pentru a realiza aceasta, se procedează astfel:

- se poziționează cursorul pe linia la care, când se ajunge, se dorește crearea primului punct;

- se apelează opțiunea **TOGGLE BREAKPOINT**, din meniul **DEBUG**, opțiune care are rolul de a marca primul punct de întrerupere (pe ecran apare o bară roșie);
- se poziționează cursorul pe următorul punct de întrerupere;
- se tastează opțiunea **TOGGLE BREAKPOINT**.

În acest fel se marchează toate punctele de întrerupere. Rularea programului se face cu **F4**. Există și alte opțiuni care permit depanarea programelor, însă considerăm că acestea sunt suficiente.

ANEXA 2.

Baze de numerație

2.1. Conversia unui număr natural din baza 10 în baza b și invers

Fie numărul natural **1354**. Acesta poate fi scris după cum se vede:

$$1354 = 1\times10^3 + 3\times10^2 + 5\times10 + 4$$

Spunem că numărul **1354** este în baza **10** (unica bază învățată până acum) și notăm acest lucru astfel: $\mathbf{1354_{(10)}}$. Baza **10** nu este singura bază posibilă. Dimpotrivă, orice număr natural **b>1** poate fi bază. De asemenea, orice număr natural poate fi scris în baza **b**.

Observăm că numărul se scrie prin utilizarea cifrelor (numere între **0** și **9**). Numărul cifrelor cu ajutorul cărora este scris un număr în baza **10** este **10** - de la **0** la **9** sunt **10** cifre.

Numărul $\mathbf{1354_{(10)}}$ poate fi scris sub forma:

$$1354 = 1\times2^{10} + 0\times2^9 + 1\times2^8 + 0\times2^7 + 1\times2^6 + 0\times2^5 + 0\times2^4 + 1\times2^3 + 0\times2^2 + 1\times2^1 + 0.$$

Acum scriem numărul în baza **2**: **10101001010** - am scris, în ordine, coeficienții puterilor bazei 2. Avem egalitatea $\mathbf{1354_{(10)} = 10101001010_{(2)}}$

Observăm că pentru scrierea unui număr în baza **2** se folosesc două cifre: **0** și **1**. Tot așa, se scrie numărul **1354** în baza **9**.

$$1354 = 1\times9^3 + 7\times9^2 + 6\times9 + 4$$

Obținem: **1764** (am scris coeficienții puterilor lui **9**) și avem egalitatea: $\mathbf{1354_{(10)} = 1764_{(9)}}$.

În exemplele anterioare numărul a fost scris în așa fel încât trecerea în baza **b** să fie imediată. Dar cum a fost obținută forma de început? De exemplu, cum a fost scris numărul **1354** ca sumă de puteri ale lui **2**? Întrebarea are sens pentru că, dacă am avea răspunsul la ea, conversia ar fi imediată. Răspunsul este complex, motiv pentru care va fi dat pe parcurs.

Pentru început, prezentăm un algoritm care ne permite să convertim numărul natural $\mathbf{n_{(10)}}$, în baza **b>1**.

- Atât timp cât **n≠0**:
  - ◊ împarte **n** la **b**.
  - ◊ în urma împărțirii, **n** reține câtul și tipărim restul.

- Numărul în baza **b** se obține scriind resturile în ordinea inversă obținerii lor.

Exemple:

1. Convertim numărul **121** din baza **10** în baza **2**.

$$121 = 2 \times \underbrace{60}_{cat} + \underbrace{1}_{rest}$$
$$60 = 2 \times \underbrace{30}_{cat} + \underbrace{0}_{rest}$$
$$30 = 2 \times \underbrace{15}_{cat} + \underbrace{0}_{rest}$$
$$15 = 2 \times \underbrace{7}_{cat} + \underbrace{1}_{rest}$$
$$7 = 2 \times \underbrace{3}_{cat} + \underbrace{1}_{rest}$$
$$3 = 2 \times \underbrace{1}_{cat} + \underbrace{1}_{rest}$$
$$1 = 2 \times \underbrace{0}_{cat} + \underbrace{1}_{rest}$$

Scriind resturile în ordine inversă avem: $\mathbf{1111001_{(2)}=121_{(10)}}$.

Probă: $1 \times 2^6 + 1 \times 2^5 + 1 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 0 \times 2 + 1 = 121$

Scriind resturile în ordine inversă obținem numărul în baza **2**: **1111001**.

2. Scriem numărul $\mathbf{121_{(10)}}$ în baza 3.

$$121 = 3 \times \underbrace{40}_{cat} + \underbrace{1}_{rest}$$
$$40 = 3 \times \underbrace{13}_{cat} + \underbrace{1}_{rest}$$
$$13 = 3 \times \underbrace{4}_{cat} + \underbrace{1}_{rest}$$
$$4 = 3 \times \underbrace{1}_{cat} + \underbrace{1}_{rest}$$
$$1 = 3 \times \underbrace{0}_{cat} + \underbrace{1}_{rest}$$

Scriind resturile în ordine inversă obținem $\mathbf{121_{(10)}=11111_{(3)}}$.

$$1 \times 3^4 + 1 \times 3^3 + 1 \times 3^2 + 1 \times 3 + 1 \times 3^0 = 81 + 27 + 9 + 3 + 1 = 121$$

3. Scriem numărul $\mathbf{121_{(10)}}$ în baza **8**.

$$121 = 8 \times 15 + 1$$
$$15 = 8 \times 1 + 7$$
$$1 = 8 \times 0 + 1$$

Scriind resturile în ordine inversă obținem: $121_{(10)}=171_{(8)}$

$$1\times 8^2+7\times 8+1=121$$

Observație: atunci când scriem un număr în baza **b**, cifrele sale sunt între **0** și **b-1**.

Exemple:

- un număr în baza **10** are cifrele cuprinse între **0** și **9**;
- un număr în baza **2** are numai cifre **0** sau **1**;
- un număr în baza **3** are cifrele cuprinse între **0** și **2**.
- un număr în baza **8** are cifrele cuprinse între **0** și **7**.

De ce? Observăm faptul că, în conversie, cifrele numărului în baza **b** sunt resturi obținute prin împărțiri succesive la **b**. La matematică am învățat faptul că *restul este întotdeauna mai mic decât împărțitorul.*

În cele ce urmează justificăm algoritmul de conversie utilizat. Așa cum rezultă din exemplele date, ideea este de a scrie numărul **N** sub forma:

$$N=a_n\times b^n+a_{n-1}\times b^{n-1}+a_{n-2}\times b^{n-2}+\ldots+a_1\times b^1+a_0,$$
$$unde, a_0,a_1,a_2,\ldots,a_n\in\{0,1,2,\ldots,b-1\}.$$

adică să găsim coeficienții a_i ai diverselor puteri ale numărului *b.* Coeficienții a_i reprezintă cifrele cu ajutorul cărora se formează numărul. Acesta este:

$$a_n a_{n-1} a_{n-2}\ldots a_1 a_0$$

Pentru a afla coeficienții a_i, procedăm ca mai jos:

$\Rightarrow$ a_0 este restul împărțirii lui **N** la **b**. Motivul?

$$N=b\times(a_n\times b^{n-1}+a_{n-1}\times b^{n-2}+a_{n-2}\times b^{n-3}+\ldots+a_1)+a_0,\ a_0\in\{0,1,2,\ldots,b-1\}$$

$\Rightarrow$ reținem câtul împărțirii lui **N** la **b** (N_1):

$$N_1=a_n\times b^{n-1}+a_{n-1}\times b^{n-2}+a_{n-2}\times b^{n-3}+\ldots+a_1;$$

$\Rightarrow$ a_1 este restul împărțirii lui N_1 la **b**

$$N_1=b(a_n\times b^{n-2}+a_{n-2}\times b^{n-3}+\ldots+a_2)+a_1.$$

$\Rightarrow$ reținem câtul împărțirii lui N_1 la **b** (N_2);

$$N_2=a_n\times b^{n-2}+a_{n-2}\times b^{n-3}+\ldots+a_2;$$

$\Rightarrow$ a_2 este restul împărțirii lui N_2 la **b**

$\Rightarrow$ . . .

$\Rightarrow$ algoritmul continuă până când câtul obținut este **0**.

Apare o problemă. Să presupunem că vrem să convertim un număr din baza **10** într-o bază **b>10**. Evident, vom avea **b** cifre. Dar care sunt ele? Noi cunoaștem numai pe cele de la **0** la **9**. Atunci?

Pentru astfel de cazuri s-a făcut o convenție: primele **10** cifre utilizate vor fi cele de la **0** la **9**, iar următoarele, în ordine, vor fi: **A** (**a**), **B** (**b**), ...

Exemplu: Fie baza 16. Vom avea cifrele:

- **0**,**1**,... ,**9**;
- **A** sau **a** pentru **10**;
- **B** sau **b** pentru **11**;
- **C** sau c pentru **12**;
- **D** sau **d** pentru **13**;
- **E** sau **e** pentru **14**;
- **F** sau f pentru **15**.

Exemplu: să scriem numărul **2809** în baza **16**.

$$2809 = 16 \times 175 + 9;$$
$$175 = 16 \times 10 + 15 \quad (F);$$
$$10 = 16 \times 0 + 10 \quad (A).$$

Obținem: $\mathbf{2809_{(10)}=AF9_{(16)}}$.

Probă: $AF9_{(16)} = 10 \times 16^2 + 15 \times 16 + 9_{(10)} = 2809_{(10)}$

2.2. Conversia unui număr subunitar pozitiv din baza 10 în baza b

Să considerăm un număr subunitar pozitiv: $\mathbf{0,675_{(10)}}$. Acesta poate fi scris și așa:

$$0{,}675 = \frac{6}{10} + \frac{7}{10^2} + \frac{5}{10^3}$$

Cum am putea izola cifrele sale? Algoritmul este următorul:

$$0{,}675 \times 10 = 6{,}75 = 0{,}75 + 6;$$
$$0{,}75 \times 10 = 7{,}5 = 0{,}5 + 7;$$
$$0{,}5 * 10 = 5 = 0 + 5;$$

Am repetat procedeul până când partea fracționară obținută este **0**.

Cum convertim un număr subunitar pozitiv (**M**) din baza **10** în baza **b**? Vom nota prin **[x]** partea întreagă a numărului **x** și prin **{x}** partea fracționară a sa. Exemplu: **[1,2]=1** și **{1,2}=0,2**.

Ideea este să scriem numărul **M** sub forma:

$$M = \frac{a_{-1}}{b} + \frac{a_{-2}}{b^2} + \frac{a_{-3}}{b^3} + \frac{a_{-4}}{b^4} + \ldots\ldots\ldots$$

și să găsim coeficienții $a_{-1}, a_{-2}, \ldots, \in \{0,1,\ldots,b\}$, apoi să scriem $0,a_{-1}a_{-2}a_{-3}\ldots$, adică numărul în baza **b**.

Cum obținem coeficienții a_{-1}?

Fie **M** numărul în baza **10**.

$a_{-1} = [M \times b]$ și $M_1 = \{M \times b\}$ pentru că:

$$M \times b = a_{-1} + \frac{a_{-2}}{b} + \frac{a_{-3}}{b^2} + \ldots$$

$$[M \times b] = a_{-1};$$

$$M_1 = \{M \times b\} = \frac{a_{-2}}{b} + \frac{a_{-3}}{b^2} + \ldots$$

$a_{-2} = [M_1 \times b]$ și $M_2 = \{M_1 \times b\}$ pentru că:

$$M_1 \times b = a_{-2} + \frac{a_{-3}}{b} + \frac{a_{-4}}{b^2} + \ldots$$

$$[M_1 \times b] = a_{-2};$$

$$M_2 = \{M_1 \times b\} = \frac{a_{-3}}{b} + \frac{a_{-4}}{b^2} + \ldots$$

Procedeul continuă până când partea fracționară este **0**, sau până scriem numărul cu precizia dorită.

Exemple:

1) Convertim în baza **2** numărul $0,625_{(10)}$.

$0,625 \times 2 = 1,250 \quad [1,250] = 1 \quad \{1,250\} = 0,25;$

$0,25 \times 2 = 0,50 \quad [0,50] = 0 \quad \{0,50\} = 0,5;$

$0,5 \times 2 = 1 \quad [1] = 1 \quad \{1\} = 0.$

Numărul obținut este $0,101_{(2)} = 0,625_{(10)}$.

Probă: $0,101_{(2)} = \frac{1}{2} + \frac{0}{2^2} + \frac{1}{2^3} = \frac{1}{2} + \frac{1}{8} = 0,625_{(10)}$.

2. Convertim în baza **2** numărul $0,9_{(10)}$.

$0,9 \times 2 = 1,8 \quad [1,8] = 1 \quad \{1,8\} = 0,8;$
$0,8 \times 2 = 1,6 \quad [1,6] = 1 \quad \{1,6\} = 0,6;$
$0,6 \times 2 = 1,2 \quad [1,2] = 1 \quad \{1.2\} = 0,2;$
$0,2 \times 2 = 0,4 \quad [0,4] = 0 \quad \{0,4\} = 0,4;$
$0,4 \times 2 = 0,8 \quad [0,8] = 0 \quad \{0,8\} = 0,8;$
$0,8 \times 2 = 1,6 \quad [1,6] = 1 \quad \{1,6\} = 0,6;$
$0,6 \times 2 = 1,2 \quad [1,2] = 1 \quad \{1,2\} = 0,2;$
$0,2 \times 2 = 0,4 \quad [0,4] = 0 \quad \{0,4\} = 0,4$

.....

Observăm că obținem un număr periodic și anume **0,1(1100)**.

3. Facem conversia în baza **16** a numărului $\mathbf{0,625_{(10)}}$.

$$0.625 \times 16 = 10,0; \quad [10] = 10_{(10)} = A_{(16)} \quad \{10\} = 0.$$

Avem $\mathbf{0,625_{(10)}=0,A_{(16)}}$.

4. Facem conversia în baza **16** a numărului $\mathbf{0,9_{(10)}}$.

$0,9 \times 16 = 14,4 \quad [14,4] = 14 = E_{16} \quad \{14,4\} = 0,4;$
$0,4 \times 16 = 6,4 \quad [6,4] = 6 = 6_{(16)} \quad \{6,4\} = 0,4;$
$0,4 \times 16 = 6,4 \quad [6,4] = 6 = 6_{(16)} \quad \{6,4\} = 0,4;$

.....

Avem $\mathbf{0,9_{(16)}=0,E(6)}$.

Observație foarte importantă. Fiind dat un număr subunitar pozitiv **M**, cu un număr finit de zecimale, la conversia sa în baza **b>1**, este posibil ca, datorită periodicității, să rezulte un număr cu un număr de zecimale infinit. Din acest motiv, algoritmul de conversie va trebui să primească ca dată de intrare numărul de zecimale dorit. În cazul în care, în urma conversiei rezultă un număr cu mai puține zecimale, se completează cu numărul corespunzător de cifre 0.

Exemplu: Conversia se face cu **8** zecimale. Atunci:

- $\mathbf{0,625_{(10)}=0,10100000_{(2)}}$.
- $0,9_{(10)} \approx 0,11100110_{(2)}$ (evident, avem o aproximație).

2.3. Legătura dintre bazele 2 și 16

Fie numărul natural $\mathbf{1011011_{(2)}}$. Îl vom scrie în baza **16**, fără să-l mai trecem prin baza 10. Cum procedăm?

- Mai întâi, separăm cifrele de la dreapta către stânga în grupe de 4.

$$\underbrace{0101}_{2}\underbrace{1011}_{1}$$

Observăm că, în cazul în care numărul cifrelor nu este multiplu de **4**, putem adăuga în față numărul de cifre necesar - dacă punem oricâte cifre de **0** în fața unui număr, valoarea sa nu se schimbă.

- Înlocuim fiecare grup de 4 cifre binare cu cifra hexa corespunzătoare (care reprezintă același număr în baza 10) și obținem:

$$\underbrace{5}_{2}\underbrace{B}_{1}$$

Am obținut numărul natural $5B_{(16)}$.

Probă: $5B_{(16)}=5*16+11=80+11=91_{(10)}$.

$1011011=1\times2^6+0\times2^5+1\times2^4+1\times2^3+0\times2^2+1\times2+1=64+16+8+2+1=91_{(10)}$

Cum de am obținut rezultatul corect?

Avem: $1\times2^6+0\times2^5+1\times2^4+1\times2^3+0\times2^2+1\times2+1=$

$0\times2^7+1\times2^6+0\times2^5+1\times2^4+1\times2^3+0\times2^2+1\times2+1=$

$(0\times2^3+1\times2^2+0\times2+1)\times2^4+1\times2^3+0\times2^2+1\times2+1=$

$5\times2^4+11=5\times16+11=5B_{(16)}$.

Raționamentul făcut pentru numărul de mai sus se poate generaliza cu ușurință, pentru fiecare număr natural.

- Invers. Fie numărul $B5_{(16)}$. Dorim să-l scriem în baza **2**. Fiecare cifră a sa va fi scrisă direct în baza **2**.

Astfel: $B_{(16)}=1011;\ 5_{(16)}=0101_{(2)};\ 5B_{(16)}=10110101_{(2)}$.

Un mecanism asemănător se poate utiliza și în cazul conversiei numerelor subunitare.

- Fie $0,1011011_{(2)}$. Pentru a-l converti în baza **16** se procedează astfel:

⇒ Se separă de la stânga către dreapta zecimalele în grupuri de **4** cifre:

$$0,\underbrace{1011}_{1}\underbrace{0110}_{2}$$

⇒ Dacă numărul cifrelor nu este multiplu de **4**, putem adăuga în dreapta numărul de cifre **0** necesar (dacă punem oricâte cifre de **0** în dreapta unui număr zecimal, valoarea sa nu se schimbă).

⇒ Se scriu cifrele hexa corespunzătoare:

$$0,\underbrace{B}_{1}\underbrace{6}_{2}$$

Se obține astfel: $0,B6_{(16)}$.

Probă:

$$0,B6_{(16)} = \frac{11}{16} + \frac{6}{16^2} = \frac{11\times16+6}{16^2} = \frac{182}{256} = \frac{91}{128}$$

$$0,1011011_{(2)} = \frac{1}{2} + \frac{0}{2^2} + \frac{1}{2^3} + \frac{1}{2^4} + \frac{0}{2^5} + \frac{1}{2^6} + \frac{1}{2^7} = \frac{2^6+2^4+2^3+2+1}{2^7} =$$

$$\frac{64+16+8+2+1}{128} = \frac{91}{128}.$$

Care este mecanismul ce ne permite să realizăm atât de simplu conversia?

$$0,1011011_{(2)} = 0,10110110_{(2)} = \frac{1}{2} + \frac{0}{2^2} + \frac{1}{2^3} + \frac{1}{2^4} + \frac{0}{2^5} + \frac{1}{2^6} + \frac{1}{2^7} + \frac{0}{2^8} =$$

$$\frac{1}{2} + \frac{0}{2^2} + \frac{1}{2^3} + \frac{1}{2^4} + \frac{1}{2^4}\times\left(\frac{0}{2} + \frac{1}{2^2} + \frac{1}{2^3} + \frac{0}{2^4}\right) = \frac{2^3+2+1}{2^4} + \frac{1}{2^4}\times\frac{2^2+2}{2^4} =$$

$$\frac{11}{16} + \frac{1}{16}\times\frac{6}{16} = \frac{11}{16} + \frac{6}{16^2} = 0,B6_{(16)}$$

Raționamentul se poate generaliza cu ușurință.

Dar care este motivul pentru care învățăm să facem conversii directe ale numerelor din baza **2** în baza **16** și invers?

Există două motive:

- Un număr, natural sau zecimal se convertește mult mai repede în baza **16** decât în baza **2**. Prin urmare, dacă se cere conversia unui număr din baza **10** în baza **2** vom converti numărul în baza **16** și pe acesta, la rândul lui, îl convertim în baza **2**. Încercați!
- Un număr în baza **16** este scris concentrat (are mai puține cifre). Din acest motiv, cu rare excepții, vom scrie numerele numai în baza **16**.

2.4. Reprezentarea numerelor reale în baza b

Reprezentarea în binar a numerelor reale se face astfel:

- *se reprezintă partea întreagă;*
- *se reprezintă partea zecimală;*
- *se scrie numărul așa cum am fost obișnuiți în baza 10;*
  - *semnul (+,-);*
  - *partea întreagă;*
  - *virgulă (sau punct);*
  - *partea zecimală.*

Exemplu: $-23,125_{(10)}$.

1. În binar.

$23_{(10)}=10111_{(2)}$;

$0,125_{(10)}=0,001_{(2)}$;

$-23,125_{(10)}= -10111,001_{(2)}$;

2. În baza 16.

$23_{(10)}=17_{(16)}$;

$0,125_{(10)}=0,2_{(16)}$;

$-23,125_{(10)}=-17,2_{(16)}$.

Probleme propuse

1. Să se convertească în bazele **2** și **16** numerele: **7**, **25**, **100**, **1000**, toate în baza **10**. Pentru fiecare conversie să se facă proba, reconvertind numărul în baza **10**.

2. Să se convertească în bazele **2** și **16** numerele din baza **10**: **0,5**, **0,25**, **0,100**, **0,625**. Pentru fiecare conversie să se facă proba reconvertind numărul în baza **10**.

3. Să se convertească în baza **2** prin conversia inițială în baza **16** numerele **1234**, **0,525**, **0,256**, **0,67**, **9,625**, **-673,89**. Pentru fiecare conversie să se facă proba reconvertind numărul în baza **10**.

4. Să se convertească în baza **2** perechile de numere de mai jos, după care să se adune în baza 2:

- **123, 87;**
- **2345, 9864;**
- **1978, 1024.**

5. Scrieți un program care citește un număr natural scris în baza **2** și tipărește numărul în baza **10**. Inițial, programul citește numărul de cifre binare. Nu se vor folosi vectori (de altfel, nici nu au fost predați). Exemplu: Dacă se citește **3** și **110**, se tipărește **6**.

6. Scrieți un program care citește un număr pozitiv scris în baza **2** și tipărește numărul în baza **10**. Inițial, programul citește numărul de cifre aflate înaintea virgulei, apoi numărul de cifre aflate după virgulă.

Exemplu: Dacă se citesc **3 2** și **110,11** se tipărește **6.75**.

7. Scrieți un program care citește un număr natural în baza **10** și tipărește numărul de cifre binare pe care le are numărul convertit în baza **2**. Exemplu: dacă se citește **8**, se tipărește **4**, pentru că $8_{(10)}=1000_{(2)}$.

8. Să se scrie un program care citește două numere naturale **n** și **b≤10**. Programul va tipări:

- **da**, dacă numărul **n** poate fi considerat în baza **b**;
- **nu**, dacă numărul **n** nu poate fi în baza **b**.

Exemple:

- **n=123, b=5**. Programul tipărește **da**.
- **n=123, b=3**. Programul tipărește **nu**.

9. Se citesc:

- **n**, număr natural;
- **a≤10**, baza în care este dat numărul **n**;
- **m**, un număr natural;
- b, baza în care este dat numărul **m**.

Programul va tipări maximul dintre **m** și **n**. Se consideră că datele sunt introduse corect.

10. Se citesc un număr natural **n** și o bază **b≤9**. Să se tipărească numărul maxim rezultat prin eliminarea unor cifre ale lui n, astfel încât acesta să poată fi în baza b.

Exemple:

- Dacă se citește **n=185** și **b=6**, programul tipărește **15**;
- Dacă se citește **n=185** și **b=9**, programul tipărește **185**;
- Dacă se citește **n=185** și **b=2**, programul tipărește **1**.

11. Se citește **n**, un număr natural. Să se tipărească cea mai mică bază în care poate fi **n**.

Exemplu: Dacă se citește **125**, se tipărește **6**.

12. Se citește **n**, un număr natural. Fie **b** baza minimă în care poate fi considerat n. Care este valoarea lui **n** în baza **10**?

13. Se citește **n**, un număr natural. Fie **b** baza minimă în care poate fi considerat **n**. Care este valoarea lui **n** în baza **10**, dacă se consideră, pe rând, în bazele **b**, **b+1**,..., **9**? Ce observați? Șirul valorilor astfel obținute este descrescător sau crescător?

14. Se citesc două numere naturale **x** și **y**. Care este baza în care trebuie să fie **x** și care este baza în care trebuie să fie **y** astfel încât diferența **x-y** să fie minimă? Dar maximă?

15. Se citesc 3 numere **a**, **b**, **c**. Se cere să se precizeze dacă există **3** baze mai mici sau egale cu **16** în care pot fi numerele **a**, **b**, **c** astfel încât **a=b+c**. În caz afirmativ se vor tipări bazele.

Exemplu: **a=10000, b=10, c=13**. Programul tipărește: **Da**.

⇒ dacă **a** este în baza **2**, atunci $10000_{(2)}=16_{(10)}$.

⇒ dacă **b** este în baza **3**, atunci $10_{(3)}=3_{(10)}$.

⇒ dacă **c** este în baza **10**, atunci $c=13_{(10)}$.

⇒ **a=b+c** pentru că **16=3+13**.

Programul tipărește **2**, **3**, **10**.

ANEXA 3

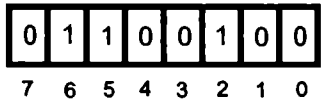
Cum se memorează datele ?

3.1. Bit, octet

Orice calculator memorează informațiile în baza **2** (succesiune de **0** și **1**). Motivul? Componentele electronice au două stări. Se face o convenție: *dacă o componentă electronică se găsește într-una din stări, se consideră că memorează 0, altfel memorează 1.* Astfel am obținut *unitatea elementară de memorare: bitul.*

Un bit reține una din două valori 0 sau 1

Un octet este alcătuit din 8 biți.

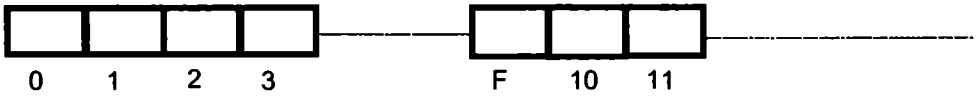


În figura de mai sus este reprezentat un octet. Acesta reține valoarea **01100100**. Mai simplu, putem nota în hexa valoarea reținută: **64**. Observăm și faptul că biții care alcătuiesc un octet sunt numerotați (de la **0** la **7**).

Astfel:

- bitul **0** reține **0**;
- bitul **1** reține **0**;
- bitul **2** reține **1**;
- . . .

Memoria internă poate fi privită ca o succesiune de octeți. Pentru a-i distinge, aceștia sunt numerotați. *Numărul de ordine al unui octet constituie adresa sa.* Adresele sunt date în binar, dar pentru a le putea scrie cu ușurință folosim baza 16.



Octetul este cea mai mică unitate de memorie direct adresabilă. Adresarea unui octet se face prin numărul său de ordine, numit adresă.

Să nu se facă confuzie între adresa unui octet și conținutul său. De exemplu, octetul cu adresa FFFF reține valoarea hexa **3B**.

Desigur, prin căi indirecte se poate accesa și bitul (chiar și în Turbo Pascal, după cum vom vedea). Aceasta nu înseamnă că bitul este direct adresabil. *Numărul de octeți ai memoriei interne dă capacitatea de memorare a acesteia.* Pentru ușurința exprimării capacității de memorare au fost introduse următoarele unități de măsură:

- Kb (Kilobyte) - 1 Kb=2^{10} octeți (bytes);
- Mb (Megabyte) - 1Mb=2^{10} Kb;
- Gb (Gigabyte) - 1Gb=2^{10} Mb.

Astfel există calculatoare cu o memorie internă de 4Mb, 8Mb, 16Mb, 32Mb, etc. Evident, este de preferat ca un calculator să dispună de cât mai multă memorie internă.

3.2. Memorarea numerelor naturale

Numerele naturale se memorează în binar. Pentru memorare, se utilizează unul sau mai mulți octeți consecutivi. Problema este următoarea: *care este cel mai mare număr care se poate memora în n poziții binare consecutive?* Evident, fiecare poziție binară trebuie să fie 1. Avem:

$$\underbrace{11111.....1111}_{de\ n\ ori}$$

Acesta este un număr binar care convertit în baza 10 este:

$$2^{n-1}+2^{n-2}+2^{n-3}+...+2^2+2+1.$$

Acum vom aplica o formulă și anume:

$$x^n-1=(x-1)(x^{n-1}+x^{n-2}+x^{n-3}+...+x^2+x+1)$$

Probă: $(x-1)(x^{n-1}+x^{n-2}+x^{n-3}+...+x^2+x+1)=x^n+x^{n-1}+x^{n-2}+...+x^2+x$

$-x^{n-1}-x^{n-2}-x^{n-3}+....-x^2-x-1=x^n-1.$

Aplicând formula de mai sus numărului convertit în baza 10 avem:

$$2^{n-1}+2^{n-2}+2^{n-3}+...+2^2+2+1=$$

$$(2-1)(2^{n-1}+2^{n-2}+2^{n-3}+...+2^2+2+1)=2^n-1.$$

Prin urmare, cel mai mare număr natural care poate fi memorat în n *poziții binare este* 2^n-1.

Exemple:

1. Un octet are 8 biți. Prin urmare, cel mai mare număr care poate fi memorat este $2^8-1=256-1=255$. Se pot memora 256 de numere (de la 0 la 255).

Fie numărul $62_{(10)}$. Îl convertim în baza 16.

Avem: $62_{(10)}=3E_{(16)}$. Apoi în baza 2 (este mai ușor așa, decât direct în baza 2): 00111110. Vom avea memorat:

00111110

Putem lăsa rezultatul în hexa (pentru ușurința citirii, nu pentru că numerele se memorează în hexa).

3E

2. Doi octeți consecutivi au **16** poziții binare. Cel mai mare număr care poate fi reținut este $2^{16}-1=2^8\times2^8-1=256\times256-1=65536-1=65535$. În concluzie, doi octeți consecutivi pot memora numere între **0** și **65535** (adică **65536** numere). Reluăm exemplul (numărul **62**). Vom avea:

00000000	00111110

00	3E

3.3. Memorarea numerelor întregi

Acestea pot fi și negative. Exemple: **123**, **-123**, **0**, **25** etc. Pentru memorarea lor se utilizează codul complementar. Să vedem cum. Pentru memorare se utilizează unul sau mai mulți octeți. Prin urmare, numărul **n**, al pozițiilor binare care reține numărul, este multiplu de **8** (numărul de biți pe care îl are un octet).

Din cele **n** poziții binare una este rezervată pentru semn. Aceasta reține:

- **0**, dacă numărul este pozitiv;
- **1**, dacă numărul este negativ.

În acest fel, rămân **n-1** poziții binare în care putem reprezenta numărul.

Dacă numărul este pozitiv, îl reprezentăm în baza 2 (utilizând cele n-1 poziții). Exemplu: Considerăm că se utilizează un octet și dorim să memorăm numărul **70**. Avem: $70_{(10)}=46_{(16)}=01000110_{(2)}$. Numărul este memorat astfel:

01000110

Pe scurt, vom nota (în baza 16):

46

Același număr, reprezentat pe doi octeți arată ca mai jos:

00000000	01000110

00	46

Observăm că *primul bit (indiferent de numărul de octeți folosit pentru reprezentare) are valoarea 0.* Ținând cont de faptul că pentru rezervare se folosesc **n-1** poziții binare, cel mai mare număr care poate fi reprezentat este $2^{n-1}-1$.

Exemple:

- Dacă folosim un octet, avem **n=8**, **n-1=7**, și cel mai mare număr care poate fi reprezentat este $2^7-1=128-1=127$.
- Dacă pentru reprezentare se folosesc 2 octeți avem: **n=16**, **n-1=15**, cel mai mare număr este $2^{15}-1=32768-1=32767$.

Dacă numărul este negativ procedăm astfel:

- *reprezentăm în baza 2 numărul cu semn schimbat, adică numărul pozitiv.*
- *valoarea memorată se obține schimbând toate cifrele 1 în 0 și toate cifrele 0 în 1 și adunând 1 (se numește complement față de 2 - vedem imediat de ce).*

Exemplu: reprezentăm pe un octet numărul **-70**.

⇒ $70_{(10)}=01000110_{(2)}$;
⇒ Inversăm cifrele: **10111001**;
⇒ La valoarea obținută, se adună **1**:

$$\begin{array}{r} 10111001+ \\ \cdot 00000001 \\ \hline 10111010 \end{array}$$

Aceasta este valoarea pe care o reprezentăm:

$$\boxed{10111010}$$

Observăm că primul bit reține **1**, deci numărul este negativ. În concluzie, bitul de semn ia valoarea care trebuie, respectând algoritmul prezentat.

Să notăm cu z_+ valoarea obținută pentru un număr pozitiv și cu z_- valoarea obținută pentru același număr, însă negativ. Avem: $z_+ + z_- = 2^n$

Exemplu: $70_{(10)}$ se reprezintă prin **01000110**;
$-70_{(10)}$ se reprezintă prin **10111010**.

Avem:
$$\begin{array}{r} 01000110+ \\ 10111010 \\ \hline 100000000 \end{array}$$

$100000000_{(2)}=2^8$ (reprezentarea s-a făcut pe **8** poziții binare).

Justificare. Cum a fost calculat z_-? Mai întâi au fost inversate cifrele **0** în **1** și **1** în **0**, apoi s-a adunat **1**. A aduna z_+ cu z_- este echivalent cu a face suma a 3 numere: z_+, numărul inversat și **1**. Dar suma primelor două numere este **11...111** (de **n** ori). Dacă adunăm acesteia numărul **1** obținem **100...0** (de **n** ori cifra **0**), adică 2^n.

Acum prezentăm formula de conversie în cod complementar. Fie **z** un număr întreg - scris în binar. Vom nota cu z^* valoarea sa în cod complementar, scrisă pe **n** poziții binare.

$$z^* = \begin{cases} z & pentru \quad z \geq 0; \\ 2^n - z & pentru \quad z < 0. \end{cases}$$

Exemplu: Să se reprezinte pe **16** poziții binare numerele **±1000**. Convertim **1000** în baza **16** (este mai ușor așa). $1000_{(10)}=3E8_{(16)}$. Se convertește valoarea obținută în baza **2**. $3E8_{(16)}=1111101000$. Reprezentăm pe **16** poziții binare rezultatul obținut:

0000001111101000	03E8

Acum reprezentăm **-1000**. Considerăm reprezentarea numărului **1000**: **0000001111101000**. Inversăm cifrele: **1111110000010111**. Adunăm **1**:

$$\begin{array}{r} 1111110000010111+ \\ 0000000000000001 \\ \hline 1111110000011000 \end{array}$$

Reprezentăm valoarea obținută în bazele **2** și **16**.

1111110000011000	FC18

Este mai ușor să aplicăm formula de trecere în cod complementar, dar lucrând în baza **16**. Astfel $2^{16}=10000_{(16)}$.

$$\begin{array}{r} 10000- \\ 003E8 \\ \hline 0FC18 \end{array}$$

Am arătat faptul că prin utilizarea codului pe **n** poziții binare, cel mai mare număr pozitiv care se poate reține este $2^{n-1}-1$. *Dar care este cel mai mic număr care poate fi reținut?*

De la început observăm că, pentru un număr negativ, prima poziție binară este **1**. Fie z_+ valoarea obținută pentru un număr pozitiv și z_- valoarea obținută pentru același număr, însă negativ ($z_++z_-=2^n$). Avem $z_-=2^n-z_+$. Dacă z ia cea mai mare valoare posibilă, z_- o ia pe cea mai mică

(descăzutul este constant iar scăzătorul este maxim, deci valoarea obținută este minimă). Cea mai mare valoare pentru **z_** este $2^{n-1}-1$. Atunci cea mai mică valoare pe care o ia **z_** este: $2^n-(2^{n-1}-1)=2\times2^{n-1}-2^{n-1}+1=2^{n-1}+1$.

Pe de altă parte, în cod complementar există posibilitatea să se rețină valoarea **10...0** (de **n-1** ori **0**). Întrucât bitul de semn este **1**, se memorează o valoare negativă, pe care o complementăm:

- inversăm cifrele: **0111...1** (de **n-1** ori **0**);
- adunăm **1** și obținem **10...0** (de **n-1** ori **0**), adică de unde am plecat;
- $10...0_{(2)}=2^{n-1}$ și dacă ținem cont că numărul este negativ, obținem -2^{n-1}.

Valoarea memorată (**10...0**) este specială (trecută în cod complementar sau invers rămâne nemodificată). S-a făcut convenția ca ea să reprezinte valoarea negativă -2^{n-1}. În concluzie, cel mai mic număr care poate fi reținut este -2^{n-1}. *Utilizând reprezentarea în cod complementar pe* **n** *poziții binare se pot memora valori întregi din intervalul:* $[-2^{n-1}, 2^{n-1}-1]$.

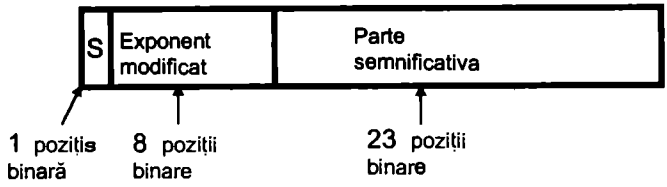
Exemple:

- Pentru un octet se memorează numere întregi cuprinse în intervalul $[-2^7, 2^7-1]=[-128, 127]$.
- Prin utilizarea a doi octeți se memorează numere întregi din intervalul $[-2^{15}, 2^{15}-1]=[-32768, 32767]$.

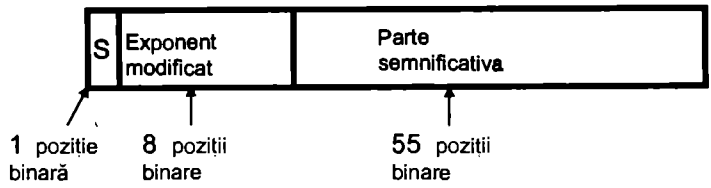
3.4. Memorarea numerelor reale

Exemple: **±2,625**, **±10,34**. *Pentru memorarea numerelor reale se utilizează* virgula mobilă. Pentru început, prezentăm schemele sub care se memorează numerele în virgulă mobilă simplă precizie și dublă precizie.

1) Virgulă mobilă simplă precizie Reprezentarea se face pe 4 octeți (32 poziții binare) sub forma:



2) Virgulă mobilă dublă precizie Reprezentarea se face pe 8 octeți (64 poziții binare) sub forma:



Avem:

a) **s** - semn. Bitul de semn este:

- **0**, dacă numărul este mai mare sau egal cu 0;
- **1**, dacă numărul este strict mai mic decât 0.

b). Exponent modificat. Este întâlnit în literatura de specialitate și sub numele de **caracteristică**.

c). Parte semnificativă. După cum observați, numărul de biți în care se memorează diferă de la reprezentarea în virgulă mobilă simplă precizie la cea în virgulă mobilă dublă precizie.

Etapele de reprezentare în virgulă mobilă a numerelor reale sunt următoarele:

- se scrie numărul în binar;
- se normalizează - adică se scrie în baza **2** sub forma $\underbrace{0,1.....}_{\text{paleta semnicativa}} \times (10)^{exp}$ (**10** reprezintă numărul $2_{(10)}$). Cu alte cuvinte, numărul se scrie de așa natură încât partea întreagă să fie **0**, iar prima cifră a părții zecimale să fie diferită de 0.
- se reprezintă conform standardului cerut. Exponentul modificat se obține după formula: Exponentul modificat = $2^7+exp_{(10)}=10000000_{(2)}+exp_{(2)}$.

Întrucât primul bit al părții semnificative este întotdeauna 1, *acesta nu va fi reținut*, dar se va ține cont de el.

- se completează bitul de semn după regula dată.

Exemple:

1. Să se reprezinte în virgulă mobilă numărul 100,75.

a. Scriem numărul în baza **2**.

$100_{(10)}=64_{(16)}$;
$0,75_{(10)}=0,C_{(16)}$;
$100,75_{(10)}=64,C_{(16)}=1101000,11_{(2)}$.

2. Se scrie numărul normalizat:

$1101000,11_{(2)}=0,110100011\times(10)^{111}$ **-111** este numărul $7_{(10)}$, adică puterea lui **2**.

3. Calculăm exponentul modificat:

$$\begin{array}{r} 10000000+ \\ \underline{00000111} \\ 10000111 \end{array}$$

4. Bitul de semn reține 0.

Numărul memorat în virgulă mobilă simplă precizie este:

$$\underbrace{0}_{}\underbrace{10000111}_{}\ \underbrace{1010001100}_{}\ \underbrace{0000000000}_{}\ \underbrace{0000}_{}$$

sau în hexa: **43D18000**.

b) Să se reprezinte în virgulă mobilă numărul **-100,75**. Am reprezentat **100,75**. Singura deosebire este că bitul de semn este **1**.

$$\underbrace{1}_{}\underbrace{10000111}_{}\ \underbrace{1010001100}_{}\ \underbrace{0000000000}_{}\ \underbrace{0000}_{}$$

sau în hexa: **C3D18000**.

Pentru a reprezenta aceleași numere în virgulă mobilă dublă precizie se adaugă în dreapta reprezentărilor obținute un număr de 32 de cifre binare 0 (sau 8 cifre hexa 0 - pentru reprezentarea respectivă).

c) Să se reprezinte în virgulă mobilă simplă precizie numărul **0,05**.

Avem $0_{(10)}=0_{(2)}$

$$0{,}05_{(10)}=0{,}00(0011)_{(2)}$$

$$0{,}05_{(2)} = 0{,}00(0011)_{(2)}=0{,}0000110011001100111...$$

$$= 0{,}11001100...\times(10)^{-100}.$$

Exponentul modificat este:

$$128_{(10)}+(-4_{(10)})=124_{(10)}=7C_{(16)}=01111100_{(2)}.$$

Partea semnificativă este: **10011001100...** (prima cifră **1** nu se reprezintă).

Reprezentarea în virgulă mobilă simplă precizie este (bitul semn reține 0):

$$\underbrace{0}_{}\underbrace{01111100}_{}\ \underbrace{1001100110}_{}\ \underbrace{0110011001}_{}\ \underbrace{100}_{}$$

În hexa, reprezentarea este: **3E4CCCCC**.

d) Să se reprezinte în virgulă mobilă simplă precizie numărul -0,05.

Pornim de la reprezentarea anterioară, cu bitul de semn **1**.

$$\underbrace{1}_{}\underbrace{01111100}_{}\ \underbrace{1001100110}_{}\ \underbrace{0110011001}_{}\ \underbrace{100}_{}$$

În hexa, reprezentarea este: **BE4CCCCC**.

Utilizând virgula mobilă simplă precizie exponentul modificat (exp+128) trebuie să verifice dubla inegalitate:

$$0 \leq exp+128 \leq 255$$

Motivele?

⇒ 255 este cel mai mare număr natural reprezentabil pe 8 poziții;
⇒ 0 este cel mai mic număr natural reprezentabil pe 8 poziții.

Din dubla inegalitate de mai sus, rezultă că exponentul trebuie să verifice relația:

$$\mathbf{-128 \leq exp \leq 127}$$

Prin urmare, cel mai mare număr care poate fi reținut în virgulă mobilă simplă precizie este:

$$0,11....1\times(10)_{(2)}{}^{1111111} = (\frac{1}{2}+\frac{1}{2^2}+...+\frac{1}{2^{23}})\times 2^{127}{}_{(10)} \equiv 1\times 2^{127} =$$
$$2^{10\times 12+7} = (2^{10})^{12}\times 2^7 = 1024^{12}\times 128 \equiv 1000^{12}\times 1,3\times 10^2 =$$
$$1,3\times(10^3)^{12}\times 10^2 = 1,3\times 10^{36}\times 10^2 = 1,3\times 10^{38}.$$

Cel mai mic număr pozitiv care poate fi reținut este:

$$0,100..0\times(10)_{(2)}{}^{-10000000} = \frac{1}{2}\times 2^{-128} = \frac{1}{2}\times\frac{1}{2^{128}} = \frac{1}{2^{129}} \equiv \frac{1}{2^{130}} \equiv \frac{1}{(2^{10})^{13}} \equiv$$
$$\frac{1}{1024^{13}} \equiv \frac{1}{(1000)^{13}} = \frac{1}{10^{39}} = 0,\underbrace{00000.....01}_{de\ 39\ ori} \equiv 0.$$

Observații:

- S-a ținut cont de faptul că bitul care reprezintă prima cifră de după virgulă este, implicit, **1** și nu se reprezintă.
- Chiar **0**, ca număr real se reprezintă aproximativ. Cu toate acestea în hexa reprezentarea sa este: **00000000** (în virgulă mobilă simplă precizie). Observația provine de la faptul că primul bit al părții semnificative este **1** și nu se reprezintă.
- Întrucât pentru reprezentarea unui număr negativ nu se modifică decât bitul de semn, este suficientă aproximarea făcută pentru numerele reale pozitive.
- Faptul că numerele reale se reprezintă aproximativ, creează o mare problemă în informatică (pentru că există aplicații practice care cer o mare precizie a calculului). Cu toate acestea se ocupă *Analiza numerică*, disciplină aflată la granița dintre informatică și matematică.
- Există mai multe standarde de reprezentare în virgulă mobilă. Ele diferă prin numărul de octeți alocați pentru reprezentare, și / sau prin numărul de biți folosiți pentru reținerea exponentului modificat și numărul de biți folosiți pentru reținerea mantisei.
- Cu cât numărul de biți folosiți pentru reprezentarea exponentului modificat este mai mare, cu atât putem reprezenta numere mai mari.
- Cu cât numărul de biți folosiți pentru reprezentarea părții semnificative este mai mare, cu atât numărul poate fi reprezentat mai precis.

- Cu cât numărul este mai mare crește probabilitatea ca acesta să fie reprezentat cu aproximație mai mare (exponentul mărit duce ca aproximația făcută la reprezentarea părții semnificative să fie mai mare).

3.5. Memorarea caracterelor

Pentru memorarea caracterelor se folosește un cod special numit ASCII. Fiecare caracter se memorează la nivelul unui octet, printr-un număr (evident între **0** și **255**) dat de codul ASCII.

Exemple:

⇒ **'a'** se memorează prin **97**;
⇒ **'b'** se memorează prin **98**;
⇒ **'c'** se memorează prin **99**;
⇒ **'A'** se memorează prin **65**;
⇒ **'B'** se memorează prin **66**;
⇒ **'C'** se memorează prin **67**;
⇒ **'0'** se memorează prin **80**;
⇒ **'1'** se memorează prin **81**;

Observații:

- Logica de codificare a caracterelor care reprezintă litere mici sau mari este de a acorda caracterelor coduri în ordine alfabetică (aceasta are implicații uriașe în programare - este permisă sortarea alfabetică);
- A nu se confunda caracterul **1** cu numărul **1**, caracterul **2** cu numărul **2** etc.

Exerciții propuse

1. Cum se memorează pe **2** octeți numerele naturale: **65**, **76**, **10000**? Dar **100000**?

2. Cum se memorează pe un octet numerele întregi **-9**, **9**, **99**, **-99**? Dar **200**?

3. Cum se memorează în virgulă mobilă simplă precizie numerele reale: **123,56**, **-123,56** **+17890,89**, **-9875**, **65464**, **-442278**?

4. Se consideră conținuturile a patru octeți consecutivi în hexa: **1AB9C135**. Ce numere sunt reprezentate?

- dacă sunt privite ca **4** numere naturale;
- dacă sunt privite ca **4** numere întregi;
- dacă sunt privite ca două numere întregi reprezentate în cod complementar, fiecare pe doi octeți;
- dacă reprezintă un număr în virgulă mobilă simplă precizie.

ANEXA 4

Exemple de utilizare a algoritmilor la fizică și chimie

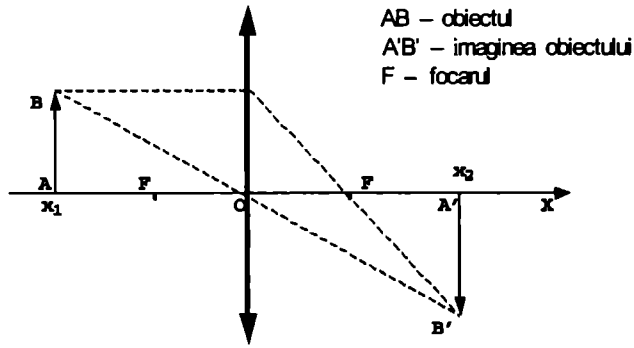
Problema lentilelor

Se consideră un șir de **n** obiecte luminoase fiecare dintre ele fiind proiectat în câte o lentilă ce este plasată în originea axei **Ox**. Cunoscându-se, pentru fiecare obiect, distanța nenulă **x** până la lentilă și distanța **y** de la lentilă până la imagine, se cere să se determine distanța focală maximă și la câte lentile este întâlnită.

Exemplu: Pentru **n=3** și perechile de valori **x=5, y=7**; **x=3, y=9**; **x=4, y=10** se va afișa:

Distanța focală maximă=**2.91** ;
Numărul de apariții=**1** ;

Rezolvare :



Problema presupune că obiectul nu se află plasat în focar, deci **x+f≠0** . Distanța focală se determină din formula lentilelor:

$$1/f = 1/x_2 - 1/x_1,$$

unde x_1 este coordonata punctului unde se află obiectul iar x_2 coordonata punctului în care se formează imaginea. Deducem că pentru fiecare obiect coordonata x_1 va fi egală cu **-x** iar coordonata x_2 cu **y**.

Pentru fiecare obiect se va determina distanța focală, actualizându-se la nevoie maximul și numărul de apariții al acestuia:

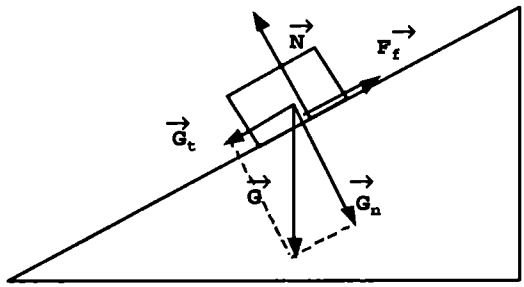
```
întreg n,i,max,nr,x,y;
Citește n;

┌pentru i←1, n execută
|  citește x, y
|  f ←(-x)*y/(-x)-y;
|  ┌dacă f=max atunci nr ← nr+1
|  |altfel ┌dacă f>max atunci
|  |       | max ← f
|  |       | nr ← 1
|  |       └■
|  └■
└■
scrie max, nr;
```

Problema planului înclinat

Pe un plan înclinat la un unghi de **45 °** se află **n** obiecte de natură diferită, numerotate de la **1** la **n**. Cunoscându-se pentru fiecare obiect distanța până la baza planului (**s**) și coeficientul de frecare **μ**, să se determine care obiect ajunge primul la baza planului și care sunt obiectele care nu se vor deplasa. (**g=10m/s²**).

Rezolvare:

Pentru fiecare obiect vom calcula timpul necesar pentru a ajunge la baza planului. Pentru aceasta trebuie determinată mărimea forței rezultante care acționează asupra obiectului.



Asupra obiectului acționează **4** forțe: greutatea normală(**G_n**), forța de frecare(**F_f**), greutatea tangențială (**G_t**) și normala la suprafață (**N**). Aplicăm principiul II al mecanicii de-a lungul planului:

$$m*a=G_t - F_f$$

Înlocuind în formulă relațiile:

1. $G = m*g$
2. $G_t = G*\sin(\alpha) = m*g*\sin(\alpha)$
3. $F_f = \mu*N = \mu*G_n = \mu*m*g*\cos(\alpha)$

Obținem accelerația: **a = g*sin(α)- μ*g*cos(α).**
Cum unghiul este de **45°** și **g** poate fi aproximat cu **10 m/s²** obținem în final valoarea accelerației:

$$a = 5\sqrt{2}(1-\mu)$$

Observație: Dacă μ > 1 atunci obiectul nu se va deplasa.
Timpul necesar ajungerii la baza planului se obține din ecuația $S=(a*t^2)/2$, adică:

$$t = \sqrt{2*s/a} = \sqrt{\frac{\sqrt{2}*s}{5(1-\mu)}}$$

Pentru a determina care obiect ajunge primul la baza planului, vom identifica numărul de ordine al obiectului pentru care se obține timpul minim.

```
întreg n, i, s, ob;
real μ, t, min;
Citește n; min ←30000;
┌pentru i←1, n execută
|  citește μ, s
|  ┌dacă μ > 1 atunci scrie 'obiectul', i, 'nu se deplaseaza'
|  | altfel
|  |    t ←√(√2*s/5(1-μ)) ;
|  |  ┌dacă t<min atunci
|  |  |  min ← t
|  |  |  ob ← i
|  |  └■
|  └■
└■
scrie min, ob;
```

Problema configurației electronice a elementelor chimice

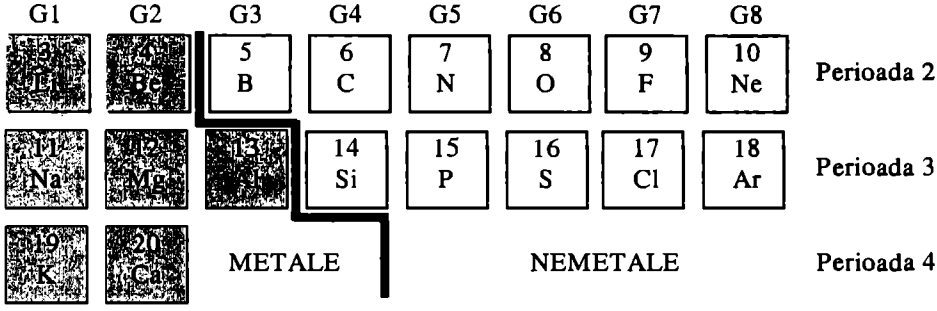
Se citește de la tastatură numărul p de protoni a n elemente chimice. Să se determine pentru fiecare element valența, natura(metal sau nemetal) și să se afișeze așezarea pe straturi a electronilor. Restricție: 2<p<21.

Exemplu: Pentru **n=3** și valorile **6, 9, 20** se va afișa :

valența **4 nemetal 2, 4**
valențele **1, 7 nemetal 2, 7**
valența **2 metal 2 8 8 2**

Rezolvare:

Elementele chimice cu număr de protoni mai mic decât 21 se află în grupele principale ale sistemului periodic a lui Mendeleev.



Algoritmul va identifica perioada și coloana elementului prin parcurgerea pe linii a sistemului. Odată găsită perioada ea va indica numărul de straturi pe care se face așezarea electronilor iar coloana va indica valența acestuia.

Pentru a afișa configurația electronică a elementului se va respecta regula: numărul maxim de electroni de pe un strat x este egal cu $2x^2$. Valența superioară indică numărul de electroni de pe ultimul strat.

Pentru a afișa natura elementului se va poziționa elementul în cadrul sistemului în funcție de perioada și grupa pe care se află.

```
întreg n,i,max,pr,p,c,j,el;
Citește n;
┌pentru el←1, n execută
| citește pr ; e ← pr; i ← 3; c ← 1; p ← 2
| ┌cât_timp i<e execută
| | ┌dacă c<8 atunci c ← c + 1
| | |altfel c ← 1
| | └■
| | ┌dacă c=1 atunci
| | | p ← p+1;
| | └■
| | i ← i + 1
| └■
| ┌dacă c<=4 atunci scrie 'valenta =', c
| |          altfel scrie 'valentele =', (8 - c), c;
| └■
```

```
max ← pr - c; i ← 2; j ← 1
cât_timp i<=p execută
  dacă max<=2*j*j atunci
          scrie max
          max ← 0
  altfel
          scrie 2*j*j
          max ← max- 2*j*j
  i ← i+1; j ← j+1
scrie c;
dacă p=2 atunci  dacă e>4 atunci scrie 'nemetal'
                 altfel scrie 'metal'
dacă p=3 atunci  dacă e>=14 atunci scrie 'nemetal'
                 altfel scrie 'metal'
dacă p=4 atunci scrie 'metal'
stop
```

ANEXA 5

Tabela codurilor ASCII

Cod Caracter	Cod Caracter	Cod Caracter	Cod Caracter	Cod Caracter	Cod Caracter
000 (nul)	022 ■ (syn)	044 ,	066 B	088 X	110 n
001 ☺ (soh)	023 ¥ (etb)	045 -	067 C	089 Y	111 o
002 ☻ (stx)	024 ↑ (can)	046 .	068 D	090 Z	112 p
003 ♥ (etx)	025 ↓ (em)	047 /	069 E	091 [	113 q
004 ♦ (eot)	026 → (eof)	048 0	070 F	092 \	114 r
005 ♣ (enq)	027 ← (esc)	049 1	071 G	093]	115 s
006 ♠ (ack)	028 ⌐ (fs)	050 2	072 H	094 ^	116 t
007 • (bel)	029 ↔ (gs)	051 3	073 I	095 _	117 u
008 _ (bs)	030 ▲ (rs)	052 4	074 J	096 `	118 v
009 □ (tab)	031 ▼ (us)	053 5	075 K	097 a	119 w
010 ■ (lf)	032 (spațiu)	054 6	076 L	098 b	120 x
011 ♂ (vt)	033 !	055 7	077 M	099 c	121 y
012 ♀ (np)	034 "	056 8	078 N	100 d	122 z
013 ♪ (cr)	035 #	057 9	079 O	101 e	123 {
014 ♫ (so)	036 $	058 :	080 P	102 f	124 |
015 ☼ (si)	037 %	059 ;	081 Q	103 g	125 }
016 ► (dle)	038 &	060 <	082 R	104 h	126 ~
017 ◄ (dc1)	039 '	061 =	083 S	105 i	127
018 ↕ (dc2)	040 (	062 >	084 T	106 j	
019 ‼ (dc3)	041)	063 ?	085 U	107 k	
020 ¶ (dc4)	042 *	064 @	086 V	108 l	
021 § (nak)	043 +	065 A	087 W	109 m	

Codul ASCII extins

Cod Caracter	Cod Caracter	Cod Caracter	Cod Caracter	Cod Caracter	Cod Caracter
128 Ç	149 ò	170 ¬	191 ┐	212 ╘	233 Θ
129 ü	150 û	171 ½	197 └	213 ╒	234 Ω
130 é	151 ù	172 ¼	193 ┴	214 ╓	235 δ
131 â	152 _	173 ¡	194 ┬	215 ╫	236 ∞
132 ä	153 Ö	174 «	195 ├	216 ╪	237 Ø
133 à	154 Ü	175 »	196 ─	217 ┘	238 ∈
134 å	155 ¢	176 ░	197 ┼	218 ┌	239 ∩
135 ç	156 £	177 ▒	198 ╞	219 ▌	240 ≡
136 ê	157 ¥	178 ▓	199 ╟	220 ▄	241 ±
137 ë	158 _	179 |	200 ╚	221 ▌	242 ≥
138 è	159 □	180 ┤	201 ╔	222 ▐	243 ≤
139 ï	160 á	181 ╡	202 ╩	223 ▀	244 ⌠
140 î	161 í	182 ╢	203 ╦	224 α	245 ⌡
141 ì	162 ó	183 ╖	204 ╠	225 ß	246 ÷
142 Ä	163 ú	184 ╕	205 ═	226 Γ	247 ≈
143 Å	164 ñ	185 ╣	206 ╬	227 π	248 °
144 É	165 Ñ	186 ║	207 ╧	228 Σ	249 •
145 æ	166 ª	187 ╗	208 ╨	229 σ	250 ·
146 Æ	167 º	188 ╝	209 ╤	230 µ	251 √
147 ô	168 ¿	189 ╜	210 ╥	231 τ	252 ⁿ
148 ö	169 _	190 ╛	211 ╙	232 ϕ	253 ²
				254 ■	255

CUPRINS

Tiparul executat la *S.C. LUMINA TIPO s.r.l.*
str. Luigi Galvani nr. 20 bis, sect. 2, Bucureşti
tel./fax 211.32.60; tel. 212.29.27
E-mail: **office@luminatipo.com**
www.luminatipo.com

Oferta completă de manuale de informatică, pentru liceu, aprobate de MEC, a Editurii L&S INFOMAT:

Tehnologia informatiei

PASCAL

Clasa a **IX**-a
Clasa a **X**-a
Clasa a **XI**-a

C++

VISUAL FOXPRO

Clasa a **XII**-a

ACCES

**Comenzile se pot face pe adresa editurii:
Str. Stânjeneilor nr. 6, bl. 30, et. 1, apt. 11, sector 4, Bucureşti sau la telefoanele:
(021) 636.63.44, Fax: (021) 332.13.15,
0722-530390, 0722-573701
e-mail: tsorin@ls-infoma.ro
web: www.ls-infomat.ro**

ISBN 973-99377-4-8

Oferta completă de manuale de informatică, pentru liceu, aprobate de MEC, a Editurii L&S INFOMAT:

Tehnologia informatiei

PASCAL

Clasa a IX-a
Clasa a X-a
Clasa a XI-a

C++

VISUAL FOXPRO

Clasa a XII-a

ACCESS

Comenzile se pot face pe adresa editurii:
Str. Stânjeneilor nr. 6, bl. 30, et. 1, apt. 11, sector 4, Bucureşti sau la telefoanele:
(021) 636.63.44, Fax: (021) 332.13.15, 0722-530390, 0722-573701
e-mail: tsorin@ls-infoma.ro
web: www.ls-infomat.ro

ISBN 973-99377-4-8