

1. Informatica și societatea

1.1. Prelucrarea informațiilor

Calculatorul a fost inventat de om pentru a prelucra informația. El îl ajută să prelucraze *foarte ușor*, într-un timp *extrem de scurt*, cu *foarte mare acuratețe*, o *mare cantitate* de informație *foarte complexă*.

Prelucrarea informației este veche de când lumea. **Prelucrarea voluntară a informației** s-a făcut însă abia atunci când babilonienii au scris primele semne cuneiforme pe tăblițele de lut. Așadar, prima manifestare a prelucrării informației a fost *scrisul*. Încă din antichitate pot fi puse în evidență două tipuri de prelucrări de informații:

- ✓ **prelucrarea textelor** = **scrisul**;
- ✓ **prelucrarea numerelor** = **calculul numeric**.

Prelucrarea automată a informației a fost posibilă o dată cu apariția calculatoarelor electronice. Așadar, scopul utilizării unui calculator este de a prelucra **informația**. Informația prelucrată poate fi formată din texte, numere, imagini sau sunete și este păstrată pe diferite medii de memorare, în diferite formate, sub formă de **date**.

Transformarea datelor în informații nu este un atribut exclusiv al calculatorului. Acest fenomen a apărut o dată cu omul. De la primele reprezentări ale unor cantități, cu ajutorul degetelor, al pietricelelor sau al bețișoarelor, și de la manipularea manuală a acestor obiecte pentru a afla câte zile mai sunt până la un anumit eveniment, sau câte animale au fost vâdate, sau câți războinici are tribul vecin, putem spune că are loc un proces de transformare a datelor în informații. Degetele, pietricelele și bețele reprezintă datele, iar ceea ce se obține prin manipularea lor (numărul de animale vâdate, numărul de zile, numărul de războinici) reprezintă informația. Ceea ce deosebește un astfel de proces de o prelucrare cu ajutorul calculatorului sunt viteza de obținere a informațiilor și modul de reprezentare a informațiilor sub formă de date.

Calculatorul nu știe să prelucraze decât șiruri de cifre binare care pot fi modelate fizic prin impulsuri de curent, cu două niveluri de tensiune, ce corespund celor două cifre binare: 0 și 1. Prin urmare, datele vor fi codificări binare ale informației existente în exteriorul calculatorului. Dacă, într-o prelucrare manuală, datele sunt reprezentate de obiecte care pot fi manipulate de om (bețișoare, pietricele sau degete), în cazul unei prelucrări automate datele vor fi reprezentate prin obiecte pe care le poate manipula calculatorul, adică șiruri de biți.

Așadar, din punct de vedere al unei prelucrări automate a informației, diferența dintre dată și informație este:

- ✓ **Informația** este un mesaj care înlătură necunoașterea unui anumit eveniment și are caracter de noutate. Informațiile sunt interpretate de oameni.

- ✓ **Data** este reprezentarea informației în interiorul calculatorului. Calculatorul nu înțelege conținutul acestor date, el numai le prelucrează prin operații specifice fiecărui tip de dată. În urma prelucrării datelor, calculatorul poate furniza omului informații.

Pentru a rezolva o anumită sarcină, trebuie să cunoaștem modul în care o putem rezolva. Pentru aceasta, trebuie să găsim o anumită **metodă**, adică un set de pași pe care trebuie să-i executăm ca să realizăm sarcina. Acest set de pași formează **algoritmul** pentru rezolvarea problemei respective. Inițial, studiul algoritmilor a fost o disciplină a matematicii prin care se căuta să se găsească un set unic de instrucțiuni prin care să se descrie rezolvarea oricărei probleme dintr-o anumită categorie. Ați învățat deja o parte din acești algoritmi: algoritmul lui Euclid pentru găsirea celui mai mare divizor comun dintre două numere, algoritmul împărțirii unui număr, algoritmul extragerii rădăcinii pătrate dintr-un număr, algoritmul conversiei unui număr reprezentat în baza zece într-un număr reprezentat într-o altă bază de numerație etc. Cu timpul, descrierea metodei de rezolvare a unei probleme cu ajutorul algoritmilor s-a extins și în alte domenii de activitate.

La fel ca și omul, și un calculator, pentru a putea rezolva o anumită sarcină, trebuie să aibă cunoștințe despre modul în care poate să o rezolve, adică să cunoască algoritmul de rezolvare a problemei. Această informație i se transmite calculatorului prin intermediul unui **program**. Deoarece **limbajul natural** (limbajul prin care comunică oamenii) nu este înțeles de calculator, care este construit astfel încât să poată prelucra numai cifre binare, programul prin care i se comunică algoritmul este scris într-un limbaj de programare. **Limbajul de programare** este un **limbaj artificial** care, prin exprimări simbolice (**instrucțiuni**), descrie operațiile de prelucrare pe care trebuie să le execute calculatorul. El îi permite omului să comunice cu calculatorul (să îi dea comenzi pe care să le execute), deoarece fiecare instrucțiune din limbajul de programare va fi tradusă într-un grup de **instrucțiuni mașină** (instrucțiuni în **limbaj mașină**), adică un șir de biți care au o anumită semnificație pentru calculator. Acest limbaj se numește limbaj mașină deoarece este propriu fiecărui tip de mașină (calculator) fiind implementat sub formă de circuite electronice în procesor.

Așadar, o sarcină se poate rezolva cu ajutorul calculatorului numai dacă modul în care se rezolvă poate fi descompus în pași pentru a putea fi descris cu ajutorul unui algoritm, deoarece calculatorul este o **mașină algoritmică**.

Dezvoltarea prelucrării automate a informațiilor cu ajutorul calculatorului s-a făcut în două direcții:

- ✓ **dezvoltarea echipamentelor** astfel încât acestea să fie capabile să stocheze cât mai multă informație, pe care să o prelucreze cu viteză cât mai mare, folosind algoritmi cât mai complecși;
- ✓ **găsirea de noi algoritmi**, cât mai performanți, pentru rezolvarea problemelor complexe și **îmbunătățirea tehnicilor de reprezentare și comunicare** a lor.

1.2. Informatica

Folosirea calculatorului a dus la apariția unei noi științe și a unui nou domeniu de activitate: **informatica**.

Informatica reprezintă un complex de discipline prin care se asigură prelucrarea rațională a informațiilor prin intermediul mașinilor automate.

Primul calculator electronic a apărut în anul 1946, ca urmare a unei cereri precise din partea armatei americane, care a fost capabilă să finanțeze un proiect atât de costisitor. Apoi, administrația americană a cumpărat primul calculator non-militar în 1951, pentru recensământul populației. Doi ani mai târziu a fost construit primul calculator destinat unei firme particulare, când General Electric a cumpărat un calculator pentru uzina sa din Louisville. Începând din 1953, firma IBM a început să pătrundă și ea pe piața de calculatoare, prezentându-și calculatoarele în mediile științifice. Astfel, calculatoarele au început să pătrundă și în mediile universitare. Dezvoltarea continuă a echipamentelor electronice de calcul a făcut ca **din 1965 informatica să nu mai fie doar o activitate anexă, ci să devină ea însăși o industrie**. În contextul unei creșteri puternice a pieței, calculatorul a devenit o unealtă folosită în toate domeniile de activitate.

La primele calculatoare electronice programele erau scrise în cod mașină (binar) sau erau cablate sub formă de circuite electronice. Modificarea unui program sau introducerea unui nou era foarte complicată, deoarece însemna introducerea programului bit cu bit. Din necesitatea rezolvării acestei probleme au apărut primele sisteme de operare și primele limbaje de programare, numite limbaje de nivel înalt. În 1956 limbajul **Fortran**, orientat pe calcule tehnico-științifice, și în 1960 limbajul **Cobol**, orientat pe aplicații economice care folosesc puține operații de calcul, dar care manipulează un volum mare de date. Limbajele de programare s-au dezvoltat continuu pentru a se adapta la noile echipamente hardware, la noile sisteme de operare și la noile cerințe ale utilizatorilor, care însemnau de fapt noi sarcini pe care trebuia să le rezolve calculatorul, adică noi algoritmi orientați pe rezolvarea anumitor probleme. În 1971 a fost creat în universitățile elvețiene limbajul **Pascal**, primul limbaj **structurat** (fiecare prelucrare elementară este considerată ca un bloc, iar blocurile pot fi închise – încapsulate – unele în altele). O dată cu apariția microcalculatoarelor, acest limbaj s-a răspândit foarte mult. Limbajul **Basic** a fost creat în Statele Unite, în 1975, ca un **limbaj interactiv** și nu putea fi folosit decât pe microcalculatoare. El permitea abordarea programării și de către persoane care nu erau specialiste în informatică. În 1971 a fost creat, de firma Bell-Telephone, limbajul **C**, pentru a permite realizarea sistemului de operare Unix. Este un limbaj foarte performant, care posedă atât conceptele limbajelor structurate de nivel înalt, cât și conceptele limbajelor de nivel scăzut, care îi permit accesul la hardware. Programele scrise în limbajele apărute recent au crescut productivitatea programatorilor. Limbajele de nivel înalt au pus bazele **ingineriei programării**.

La începutul anilor '60, în mediile universitare au început să se formeze departamente pentru cercetarea și studierea calculatoarelor. Cu timpul, a apărut o bogată literatură de specialitate, iar cursurile din domeniul informaticii au început să fie orientate pe subdomenii și să fie gradate pe niveluri de dificultate. Astăzi, **informatica este divizată în nouă subdomenii**:

1. **Algoritmi și structuri de date.** Studiază metodele prin care se pot obține aplicații care să prelucreze diferite clase de informații, modul în care vor fi reprezentate informațiile care vor fi prelucrate și metodele de optimizare a pașilor necesari pentru realizarea aplicațiilor. Scopul acestui subdomeniu este de a identifica problemele care pot fi descrise cu ajutorul algoritmilor, de a găsi modul în care trebuie procedat pentru a descoperi algoritmul și metodele de analiză și comparare a caracteristicilor algoritmilor pentru a obține algoritmi cât mai eficienți.
2. **Limbaje de programare.** Studiază notațiile (limbajele) prin care vor fi reprezentați algoritmi și structurile de date, astfel încât aplicația să poată fi prelucrată. Aceste limbaje sunt apropiate limbajului natural și pot fi ușor traduse în secvențe de comenzi pe care să le înțeleagă calculatorul. Scopul acestui subdomeniu este de a găsi noi tehnici de reprezentare și comunicare a algoritmilor.
3. **Arhitectura calculatoarelor.** Studiază modul în care sunt organizate diferite componente hardware ale calculatorului și modul în care sunt conectate pentru a putea obține un sistem eficient, sigur și util. Scopul acestui subdomeniu este de a obține mașini algoritmice cât mai bune folosind cunoștințele despre algoritmi dobândite și tehnologia existentă.
4. **Sisteme de operare.** Studiază felul în care trebuie să fie organizate programele care controlează și coordonează toate operațiile din sistemul de calcul. Scopul acestui subdomeniu este de a face un calculator să rezolve în același timp mai multe sarcini, fără ca pașii algoritmilor care descriu rezolvarea acestor sarcini să interfereze unii cu alții, iar atunci când este cazul să se poată realiza comunicarea între diverși algoritmi.
5. **Ingineria programării.** Studiază metodele prin care poate fi automatizată activitatea de proiectare a aplicațiilor, de prelucrare a informațiilor, astfel încât să se obțină programe corecte, eficiente, fără erori și ușor de exploatat.
6. **Calcul numeric și simbolic.** Studiază descrierea fenomenelor din lumea reală prin intermediul formulelor matematice, care pot fi manipulate algebric astfel încât să se obțină modele matematice ușor de descris prin algoritmi. Scopul acestui subdomeniu este de a găsi modele matematice care să permită descrierea și reprezentarea în calculator a fenomenelor complexe, cum sunt: zborul avioanelor, curenții marini, traiectoria sateliților și a planetelor, mișcarea particulelor etc.
7. **Sisteme de gestiune a bazelor de date.** Studiază modul în care pot fi organizate cantități mari de date ce nu necesită în prelucrare calcule matematice complexe. Este cazul informațiilor prelucrate în procesele economico-sociale, în întreprinderi și în administrație. Prelucrarea acestor date trebuie să se facă eficient, fără erori, cu asigurarea securității lor.
8. **Inteligența artificială.** Studiază modul în care percepe și raționează mintea umană cu scopul de a putea fi automatizate aplicații pe care omul le realizează prin metode „inteligente”, care sunt dificil de descris cu ajutorul algoritmilor, ca de exemplu înțelegerea unui limbaj, crearea de noi teorii matematice, compunerea muzicii, crearea unei opere de artă, luarea unor decizii în urma evaluării unor situații complexe (stabilirea unui diagnostic în medicină, mutarea pieselor la jocul de șah etc.).

9. **Animatie și robotică.** Studiază metodele prin care pot fi generate și prelucrate imaginile și modul în care se poate răspunde unei situații din exterior prin acționarea unui robot.

1.3. Etapele rezolvării unei probleme

Orice prelucrare automată a informațiilor presupune definirea următorului lanț:



Din această cauză, pentru orice rezolvare a unei probleme cu ajutorul calculatorului trebuie parcurse următoarele etape:

1. **analiza problemei;**
2. **elaborarea modului de rezolvare a problemei;**
3. **codificarea modului de rezolvare a problemei într-un limbaj de programare;**
4. **testarea programului și corectarea erorilor.**

Analiza problemei. Această etapă constă în formularea enunțului problemei, din care vor rezulta **specificațiile** complete și precise ale programului care va rezolva problema. Aceste specificații trebuie să țină cont de condițiile concrete de realizare a programului. Specificațiile sunt:

- ✓ **Funcția programului.** Prin ea se determină ceea ce urmează să realizeze programul.
- ✓ **Identificarea fluxului de informații.** Aceasta presupune identificarea **informațiilor de intrare** și, respectiv, a **informațiilor de ieșire** care vor fi descrise cu ajutorul **datelor**: **date de intrare** și, respectiv, **date de ieșire**.

Fiecare tip de informație îi corespunde un anumit mod de stocare în mediul de memorare, adică un anumit tip de dată. Între datele prelucrate de un program există diferite relații. Modul în care vor fi aranjate aceste date în mediul de memorare depinde de legătura dintre ele.

Elaborarea modului de rezolvare a problemei. Această etapă constă în găsirea metodei prin care să se poată rezolva problema. Ea presupune identificarea **prelucrărilor** care se fac asupra datelor de intrare pentru a obține datele de ieșire. Descrierea acestor prelucrări se face cu ajutorul **algoritmului** de rezolvare a problemei. Această fază este cea mai importantă și cea mai grea, deoarece presupune definirea logică a unei secvențe de operații pe care să le poată executa calculatorul astfel încât să se obțină rezultatele dorite.

Codificarea modului de rezolvare a problemei într-un limbaj de programare. Algoritmul de rezolvare a problemei este transpus într-un limbaj de programare ales în conformitate cu specificul problemei care trebuie rezolvată, pentru a fi comunicat calculatorului.

Testarea programului și corectarea erorilor. Pentru testarea programului se va folosi o mulțime de seturi de date de intrare care trebuie să prevadă toate situațiile care pot să apară în exploatarea curentă a programului. Testarea constă în executarea repetată

a programului pentru fiecare set de date de intrare. Dacă această mulțime de seturi de date nu este aleasă corect, programul nu va fi testat pe toate traseele algoritmului și în etapa de exploatare pot apărea erori. În această etapă se pun în evidență erorile de sintaxă, erorile de logică și dacă reprezentarea externă a rezultatelor are aspectul grafic dorit. Erorile de sintaxă apar din scrierea incorectă a instrucțiunilor și ele vor fi corectate în program. Erorile de logică apar din cauza metodei de rezolvare alese și ele vor trebui identificate în cadrul algoritmului și corectate în program.

Așadar, pentru ca un calculator **să poată produce informații**, trebuie ca, la rândul său, **să primească două categorii de informații**:

- ✓ Descrierea modului în care să realizeze sarcina, adică **algoritmul**, care i se comunică sub forma unui **program**.
- ✓ Informațiile de care are nevoie algoritmul ca să realizeze acea sarcină, care i se comunică sub formă de **date de intrare**.

Studiu de caz

Scop: exemplificarea etapelor de rezolvare a unei probleme.

Enunțul problemei: Fiind date două numere reale a și b , să se rezolve ecuația de gradul întâi cu acești coeficienți: $ax+b=0$.

În urma analizei problemei se obține **specificarea programului**:

- ✓ **Funcția programului.** Dacă pentru ecuația de gradul întâi $ax+b=0$ există o soluție reală, se calculează, în caz contrar se afișează un mesaj.
- ✓ **Informațiile de intrare** sunt coeficienții ecuației, iar suportul extern prin care se vor introduce este tastatura. Reprezentarea internă a informației se va face prin **datele de intrare** a și b .
- ✓ **Informația de ieșire** va fi soluția ecuației, dacă există, iar dacă nu există, un mesaj. Suportul extern pe care va fi reprezentată informația de ieșire este ecranul monitorului. Reprezentarea internă a soluției ecuației se va face prin **data de ieșire** x .

Metoda folosită pentru rezolvarea problemei va fi algoritmul matematic de rezolvare a ecuației de gradul întâi.

Pentru testarea programului se va considera că un set de date de intrare este format de perechea de coeficienți $(a;b)$, iar o mulțime completă de seturi de date de intrare poate fi $\{(0; 0), (0; 1.5), (2.5; 1.5)\}$.



1.4. Algoritmul

Datele de intrare sunt supuse unui proces de prelucrare pentru a se obține datele de ieșire. În funcție de rezultatele care se doresc, prelucrarea datelor este realizată după un anumit algoritm.

Algoritmul reprezintă o mulțime ordonată și finită de pași executabili prin care se definește fără echivoc modul în care se poate realiza o anumită sarcină.

Informatica

Între datele de intrare și datele de ieșire ale algoritmului există o relație bine determinată de însăși construcția algoritmului.

În activitățile zilnice întâlnim la tot pasul algoritmi: **algoritmul de utilizare a mașinii de spălat rufe sau vase** (exprimat prin setul de instrucțiuni din cartea tehnică a mașinii sau de pe capacul mașinii de spălat), **algoritmul de înregistrare pe o casetă video** (exprimat prin setul de instrucțiuni din cartea tehnică a videorecorderului), **algoritmul de interpretare a muzicii** (exprimat prin partitură), **algoritmul de construire a unui model de avion sau de navă** (exprimat prin setul de instrucțiuni care însoțesc piesele care compun modelul), **algoritmul de rezolvare a unei probleme matematice** (exprimat printr-un set unic de operații prin care se descrie modul de rezolvare a oricărei probleme dintr-o categorie de probleme). De fapt, aproape toate acțiunile noastre se desfășoară după un algoritm bine definit. Un exemplu de algoritm al activităților zilnice este o convorbire telefonică:

- Pasul 1.** Început.
- Pasul 2.** Mergi la telefon.
- Pasul 3.** Ridică microreceptorul telefonului.
- Pasul 4.** Dacă are ton, formează numărul de telefon; altfel, pleacă la vecin și mergi la Pasul 10.
- Pasul 5.** Dacă telefonul este ocupat, închide telefonul și mergi la Pasul 11; altfel, așteaptă să răspundă.
- Pasul 6.** Dacă nu răspunde, pune microreceptorul în furcă și mergi la Pasul 12; altfel, începi discuția cu persoana care a răspuns.
- Pasul 7.** Dacă a răspuns persoana căutată, mergi la Pasul 9; altfel, cere să vină la telefon persoana căutată.
- Pasul 8.** Dacă persoana căutată nu poate să vină la telefon, mergi la Pasul 13; altfel, așteaptă să vină la telefon.
- Pasul 9.** Discută la telefon cu persoana căutată și mergi la Pasul 13.
- Pasul 10.** Anunță la serviciul „Deranjamente telefoane” că ai telefonul defect și mergi la Pasul 14.
- Pasul 11.** Așteaptă 15 minute și mergi la Pasul 2.
- Pasul 12.** Așteaptă 1 oră și mergi la Pasul 2.
- Pasul 13.** Închide telefonul.
- Pasul 14.** Terminat.

Un exemplu de algoritm matematic este rezolvarea ecuației de gradul întâi:

$$a \times z + b = 0$$

unde **a** și **b** sunt coeficienții ecuației și pot lua orice valori din domeniul numerelor reale, iar **z** reprezintă un număr care se calculează și care poate lua și el orice valoare reală, astfel încât să fie îndeplinită relația definită prin ecuație. Algoritmul de rezolvare a ecuației va prezenta un set unic de operații prin care se calculează valoarea lui **z** oricare ar fi valorile pentru **a** și **b**:

- Pasul 1.** Început.
- Pasul 2.** Comunică valorile pentru **a** și **b**.
- Pasul 3.** Compară **a=0**. Dacă este adevărat, execută Pasul 4; altfel, execută Pasul 7.

- Pasul 4.** *Compară $b=0$. Dacă este adevărat, execută Pasul 5; altfel, execută Pasul 6.*
- Pasul 5.** *Comunică mesajul "Ecuația are o infinitate de soluții". Mergi la Pasul 9.*
- Pasul 6.** *Comunică mesajul "Ecuația nu are soluții". Mergi la Pasul 9.*
- Pasul 7.** *Calculează $z=-b/a$.*
- Pasul 8.** *Comunică valoarea lui z .*
- Pasul 9.** *Terminat.*

Numărul de pași este finit (9 pași). Toți pașii reprezintă acțiuni care se pot executa: compară, calculează, comunică. O dată definit acest algoritm, pașii lui se vor executa pentru orice valori ale lui a și b , deci algoritmul descrie rezolvarea unei probleme generale. La fiecare executare a algoritmului care descrie o problemă generală va fi tratat un caz particular, adică se rezolvă ecuația de gradul întâi pentru valori precizate ale lui a și b , ca de exemplu $2x - 4 = 0$ ($a = 2$, $b = -4$) sau $0x - 4 = 0$ ($a = 0$, $b = -4$) sau $0x - 0 = 0$ ($a = 0$, $b = 0$).

Așadar algoritmi au următoarele **proprietăți**:

- ✓ **Claritatea.** Orice algoritm trebuie să fie precis definit, să prezinte clar toate etapele care trebuie parcurse până la obținerea soluției, fără să formuleze nimic ambiguu.
- ✓ **Finitatea.** Algoritmul trebuie să fie format dintr-un număr finit de pași, prin executarea cărora să se ajungă la rezolvarea problemei și obținerea rezultatelor.
- ✓ **Sucesiunea determinată a pașilor.** Pașii care compun algoritmul trebuie executați într-o ordine bine determinată. De obicei ei se execută în ordine secvențială (ordinea în care au fost scriși). În cazul în care apare necesitatea schimbării acestei ordini, trebuie să se precizeze clar pasul care urmează să fie executat.
- ✓ **Universalitatea.** Algoritmul trebuie să permită rezolvarea unei clase de probleme, care sunt de același tip și care diferă între ele numai prin datele de intrare. El trebuie să ofere posibilitatea de a rezolva orice problemă din acea clasă de probleme.
- ✓ **Realizabilitatea.** Pașii care compun algoritmul trebuie să reprezinte operații care se pot executa cu resursele disponibile.
- ✓ **Eficiența.** Operațiile care compun algoritmul trebuie alese astfel încât soluția problemei să fie obținută după un număr minim de pași, cu precizia prestabilită sau cu o precizie satisfăcătoare.

Evaluare

Răspundeți:

1. Ce este un algoritm? Ce sunt pașii algoritmului?
2. Determinați algoritmul pentru prepararea unui ceai. Identificați proprietățile algoritmului, în acest exemplu.
3. Citiți o rețetă din cartea de bucate. Determinați algoritmul pentru prepararea produsului culinar.
4. Dați patru exemple de probleme a căror rezolvare nu poate fi descrisă cu ajutorul algoritmului și patru exemple de probleme a căror rezolvare poate fi descrisă cu ajutorul algoritmului.

5. Prin definiție, calculatorul este o unealtă care ajută omul să execute mai bine și mai ușor unele sarcini. Ce gen de sarcini poate executa calculatorul? În ce domenii poate fi folosit calculatorul pentru a ajuta oamenii pentru realizarea acestor sarcini?
6. De ce calculatorul este o mașină care prelucrează informația? Enumerați patru motive care să justifice acest răspuns.
7. Aplicațiile care presupun calcule complexe executate repetat, precum și cele care necesită alcătuirea de tabele, au constituit motivația apariției și dezvoltării calculatoarelor. Dați un exemplu pentru fiecare dintre aceste aplicații și explicați cât de greu i-ar fi omului să execute aceste operații fără ajutorul unui echipament de calcul electronic. Dați exemple de activități din liceu în care poate fi folosit calculatorul.
8. Ce legături există între calculator și matematică? Dar între informatică și matematică? Numiți subdomeniile informaticii în care aceste legături sunt foarte importante.
9. Nu toate aplicațiile de prelucrare a informațiilor pot fi automatizate folosind un calculator. Exemplificați cu trei genuri diferite de aplicații în care folosirea calculatorului este limitată.
10. Dați două exemple în care executarea unei aplicații cu ajutorul calculatorului devine mai dificilă decât executarea ei manuală.

Alegeți:

1. Algoritmul prin care s-a descris rezolvarea ecuației de gradul întâi folosește pași prin care se execută operații de comunicare, comparație și calcul. Această proprietate a algoritmului se numește:
a) claritate b) realizabilitate c) eficiență
2. Algoritmul prin care s-a descris rezolvarea ecuației de gradul întâi permite obținerea soluțiilor pentru orice combinație de valori ale coeficienților a și b . Această proprietate a algoritmului se numește:
a) finitate b) claritate c) universalitate

Rezolvați:

1. Se consideră următorul enunț: *Fiind dat un număr a care reprezintă lungimea laturii unui pătrat, să se calculeze perimetrul, aria și diagonala pătratului.* Descrieți etapele de rezolvare a acestei probleme cu ajutorul calculatorului.
2. Se consideră următorul enunț: *Fiind date trei numere a , b și c să se verifice dacă pot reprezenta lungimile laturilor unui triunghi și, în caz afirmativ, să se calculeze aria triunghiului.* Descrieți etapele de rezolvare a acestei probleme cu ajutorul calculatorului.
3. Se consideră următorul enunț: *Fiind date trei numere a , b și c să se verifice dacă ele pot forma o progresie geometrică.* Descrieți etapele de rezolvare a acestei probleme cu ajutorul calculatorului.

2. Datele

2.1. Definiția datelor

Datele sunt obiecte prelucrate de algoritm.

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

Din punct de vedere logic, data este definită printr-o tripletă:

data elementară = (identificator, valoare, atribute)

Identificatorul datei

Identificatorul datei este un nume format din unul sau mai multe caractere și este atribuit unei date de către cel care definește data, pentru a o putea distinge de alte date și pentru a putea face referiri la ea în procesul de prelucrare a datelor. De exemplu *alfa*, *a11* sau *b1_1*.

Fiecare limbaj de programare are implementat diferit conceptul de identificator al datei, practicând constrângeri pentru:

- ✓ **numărul maxim de caractere din nume** (de exemplu, 10 caractere în cazul limbajului programare Visual Basic și 64 de caractere în cazul limbajului de programare Pascal),
- ✓ **caracterele acceptate în nume** (de exemplu, majoritatea limbajelor de programare nu acceptă în nume decât cifre, litere și caracterul linie de subliniere), și
- ✓ **caracterul care poate fi folosit la începutul numelui** (de exemplu, marea majoritate a limbajelor de programare nu acceptă ca numele unei date să înceapă cu o cifră).

De aceea, atunci când învățați să definiți și să manipulați date folosind un anumit limbaj de programare, trebuie să aflați ce constrângeri există pentru identificatorul datei.

Valoarea datei

Valoarea datei reprezintă conținutul zonei de memorie în care este stocată data. Se definește ca **domeniu de definiție al datei** mulțimea valorilor pe care le poate lua data în procesul de prelucrare.

Atributele datei

Atributele sunt proprietăți ale datelor care determină modul în care sistemul va trata datele. Cel mai important atribut este **tipul datei**.

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

Indiferent de tipul de date ales, reprezentarea datei în memoria calculatorului se face printr-un șir de biți. Pentru a realiza această reprezentare, fiecare limbaj de programare are implementați **algoritmi de codificare** care asigură corespondența dintre tipul de dată și șirul de biți, atât la scrierea datelor, cât și la citirea lor.

Așadar, **orice sistem care prelucrează informația sub formă de date trebuie să aibă definit clar conceptul de dată**. Definirea conceptului de dată implică definirea următoarelor elemente:

- ✓ cum poate fi identificată data?
- ✓ cum va fi reprezentată data în memoria calculatorului?
- ✓ ce proprietăți are data?
- ✓ cum pot fi grupate datele în colecții de date?

2.1.1. Clasificarea datelor

Clasificarea datelor se poate face folosind mai multe criterii:

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

Clasificarea în funcție de momentul în care se produc

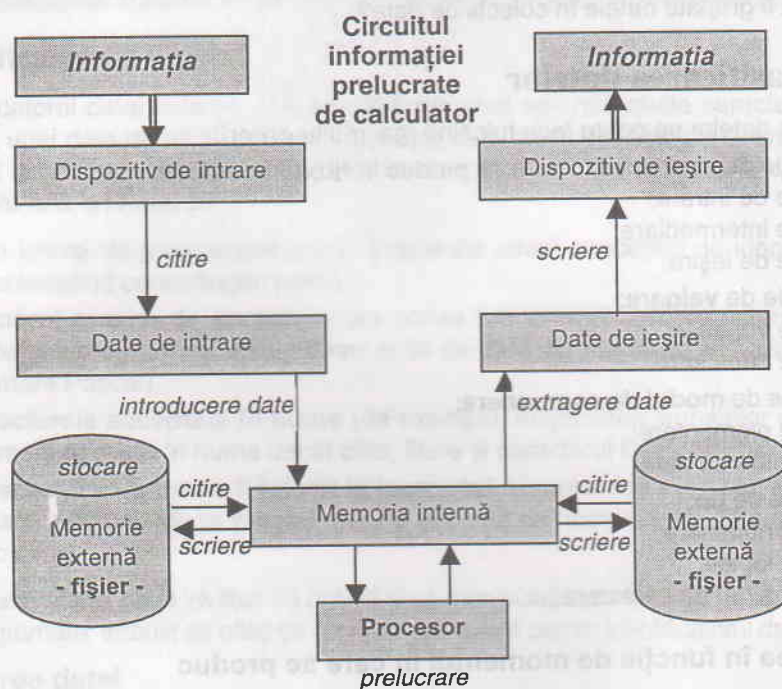
Datele se clasifică în:

- ✓ **Date de intrare.** Ele reprezintă datele care urmează să fie prelucrate în cadrul algoritmului. Sunt folosite pentru a descrie diverse evenimente și sunt informații produse în urma realizării unui eveniment, adică evaluări neprelucrate ale aceluiași eveniment. De exemplu, pot fi date de intrare notele unui elev. Pentru a putea fi prelucrate de procesor, datele sunt introduse în **memoria internă** a calculatorului. Introducerea datelor se face prin intermediul unor echipamente speci-

alizate în **citirea** informației, numite **dispozitive de intrare** (tastatură, scanner, creion optic etc.). **Dispozitivul standard de intrare este tastatura.**

- ✓ **Date de ieșire.** Ele sunt folosite pentru a descrie rezultatele obținute în urma prelucrărilor din cadrul algoritmului și furnizează informațiile pentru care a fost realizat algoritmul, ca de exemplu mediile semestriale și anuale ale elevului. Datele de ieșire sunt produse de procesor în urma operației de prelucrare și sunt depuse în memoria internă. Pentru a fi vizualizate de om, ele sunt extrase din memoria internă prin intermediul unor echipamente specializate în **scrierea** informației, numite **dispozitive de ieșire** (ecran, imprimantă etc.). **Dispozitivul standard de ieșire este ecranul.**
- ✓ **Date intermediare sau de manevră.** Ele sunt folosite în cadrul algoritmului pentru realizarea unor prelucrări.

În vederea prelucrării, datele pot fi păstrate temporar în memoria internă sau în memoria externă (discul flexibil, hard-discul, discul compact etc.). Operația se numește **stocarea** datelor.



În memoria externă datele sunt păstrate în fișiere. **Un fișier este o colecție de date organizate ca o singură unitate.** Dacă datele sunt păstrate în **fișiere**, ele vor putea fi folosite ulterior ca date de intrare într-un alt algoritm.

Așadar, orice rezolvare de problemă începe prin definirea datelor, continuă cu prelucrarea lor în conformitate cu algoritmul folosit și se termină fie cu afișarea valorii lor, fie cu stocarea lor pe un mediu de memorare în vederea prelucrării lor ulterior.

Studiu de caz

Scop: exemplificarea tipurilor de date care pot să apară într-un algoritm.

Enunțul problemei: Să se calculeze media aritmetică a n numere întregi introduse de la tastatură.

În urma analizei problemei se obține **specificația programului**:

- ✓ **Funcția programului.** Se calculează suma numerelor introduse de la tastatură și se împarte la numărul de elemente n .
- ✓ **Informațiile de intrare** sunt: numărul de elemente ale mulțimii de numere și numerele care se citesc. Reprezentarea internă a informației se va face prin **datele de intrare**: n pentru numărul de numere și a pentru un număr citit curent.
- ✓ Pentru operațiile executate în cadrul algoritmului se vor folosi **datele intermediare suma** în care se calculează suma numerelor introduse de la tastatură, prin intermediul datei de intrare a , și i prin care se numără câte numere s-au introdus de la tastatură la un moment dat. Data intermediară i este necesară pentru a afla când se termină procesul de introducere a celor n numere de la tastatură, având funcția unui contor. Valoarea inițială a sumei și a contorului (înainte de a se citi primul număr) este 0.
- ✓ **Informația de ieșire** va fi media aritmetică a celor n numere. Reprezentarea internă a mediei aritmetice se va face prin **data de ieșire media** a cărei valoare se calculează prin împărțirea sumei calculate cu ajutorul datei intermediare **suma** la numărul de elemente memorat în **data de intrare n**.

Așadar, datele folosite pentru rezolvarea acestei probleme sunt:

- ✓ **Date de intrare:** n și a .
- ✓ **Date intermediare:** **suma** și i .
- ✓ **Date de ieșire:** **media**.

iar algoritmul de rezolvare a problemei va prezenta un **set unic de operații** prin care se calculează valoarea mediei, oricare ar fi numărul de numere și valorile lor.

Pasul 1. Început.

Pasul 2. Comunică valoarea pentru n .

Pasul 3. Atribue valorile inițiale datelor **suma** și i : **suma=0** și **i=0**.

Pasul 4. Compară $i \leq n$. Dacă este adevărat, execută **Pasul 5**; altfel, execută **Pasul 8**.

Pasul 5. Comunică valoarea pentru a .

Pasul 6. Calculează **suma=suma+a** (adună la sumă noua valoare a lui a).

Pasul 7. Calculează **i=i+1** (crește contorul i cu 1 deoarece s-a mai citit un număr a). Mergi la **Pasul 4**.

Pasul 8. Calculează **media=suma/n**.

Pasul 9. Comunică valoarea datei **media**.

Pasul 10. Terminat.

Observații:

1. Pașii care conțin acțiuni de comunicare folosesc numai date de intrare și de ieșire, nu și date intermediare. Datele intermediare apar numai în pași care conțin acțiuni de calcul, de atribuire sau de comparare.
2. Valoarea datelor de ieșire se calculează în cadrul algoritmului și se comunică printr-o operație de scriere. Pentru calcularea datelor de ieșire se folosesc date de intrare și/sau date intermediare. Aceste date trebuie să aibă o valoare înainte de a fi folosite în pașii care conțin acțiuni de calcul. Valoarea datelor de intrare este comunicată prin operația de citire de la tastatură. Datelor intermediare li se atribuie o valoare inițială în cadrul algoritmului.
3. Orice nouă operație de citire executată cu o dată de intrare distruge vechea valoare memorată în dată.

**Clasificarea în funcție de valoare**

Datele se clasifică în:

- ✓ **Date variabile sau variabile de memorie.** Pe parcursul procesului de prelucrare, valoarea acestor date se poate modifica, în limitele domeniului de definiție. În timpul execuției programului ele pot avea o valoare inițială, mai multe valori intermediare și o valoare finală. În exemplul precedent pentru calcularea mediei a n numere introduse de la tastatură, data *suma* are o valoare inițială 0, mai multe valori intermediare, câte una pentru fiecare operație de citire a unui număr a , și o valoare finală, obținută după ce s-au citit toate cele n numere.
- ✓ **Date constante sau constante.** Pe tot parcursul procesului de prelucrare, data își va păstra aceeași valoare din domeniul de definiție al datei.

Studiu de caz

Scop: exemplificarea tipurilor de date care pot să apară într-un algoritm.

Enunțul problemei 1: Să se calculeze aria pentru n cercuri, fiecare cerc având o rază precizată r .

În urma analizei problemei se obține **specificația programului**:

- ✓ **Funcția programului.** Se calculează aria pentru n cercuri folosind formula matematică $aria = \pi \times r^2$, unde r este raza unuia dintre cele n cercuri.
- ✓ **Informațiile de intrare** sunt: numărul de cercuri, razele cercurilor și valoarea numărului π . Reprezentarea internă a informației se va face prin **datele de intrare**: n pentru numărul de cercuri, r pentru raza unui cerc, și π pentru numărul π .
- ✓ Pentru operațiile executate în cadrul algoritmului se va folosi **data intermediară** i care reprezintă numărul cercului pentru care se calculează aria. Această dată intermediară este necesară pentru a afla când se termină procesul de calculare a ariei celor n cercuri și are valoarea inițială 0 (nu s-a calculat aria nici unui cerc).

- ✓ **Informațiile de ieșire** vor fi ariile celor n cercuri. Reprezentarea internă a ariei se va face prin **data de ieșire** aria a cărei valoare se calculează prin formula matematică. $aria = \pi \times r^2$.

Algoritmul de rezolvare a problemei va prezenta un set unic de operații prin care se calculează valoarea ariei, oricare ar fi numărul de cercuri și valoarea razelor lor:

- Pasul 1.** Început.
- Pasul 2.** Comunică valoarea pentru n .
- Pasul 3.** Atribue valoarea inițială contorului i : $i=0$.
- Pasul 4.** Compară $i \leq n$. Dacă este adevărat, execută Pasul 5; altfel, execută Pasul 9.
- Pasul 5.** Comunică valoarea pentru r .
- Pasul 6.** Calculează $aria = \pi \times r^2$.
- Pasul 7.** Comunică valoarea ariei pentru cercul i .
- Pasul 8.** Calculează $i=i+1$ (crește contorul i cu 1 deoarece s-a citit raza unui cerc r). Mergi la Pasul 4.
- Pasul 9.** Terminat.

Datele n , i , arie și r sunt date variabile. Valoarea datei n se modifică pentru fiecare execuție a algoritmului (corespunde valorii introduse de la tastatură). Valoarea datelor i , r și arie se modifică în funcție de numărul cercului pentru care se citește raza și se calculează aria în timpul execuției algoritmului. Data π este o dată constantă. Indiferent de datele de intrare cu care se execută algoritmul, valoarea ei este aceeași și corespunde valorii numărului π .

Enunțul problemei 2: Să se afișeze numerele pare care au două cifre.

În urma analizei problemei se obține **specificația programului**:

- ✓ **Funcția programului.** Se generează numerele pare dintr-un interval $[a,b]$.
- ✓ **Informațiile de intrare** sunt limitele intervalului. Deoarece din enunțul problemei rezultă că $a=10$ (primul număr par cu două cifre) și $b=98$ (ultimul număr par cu două cifre), reprezentarea internă a informației se va face prin **datele de intrare constante**: 10 și 98. Deoarece aceste date au valori constante, ele nu trebuie citite de la tastatură.
- ✓ **Informația de ieșire** vor fi numerele pare din intervalul precizat. Reprezentarea internă a unui număr par se va face prin **data de ieșire** n a cărei valoare se calculează prin incrementarea cu 2 a valorii anterioare: $n=n+2$.

Algoritmul de rezolvare a problemei va fi:

- Pasul 1.** Început.
- Pasul 2.** Atribue valoarea inițială numărului n : $n=10$.
- Pasul 3.** Compară $n \leq 98$. Dacă este adevărat execută Pasul 4, altfel execută Pasul 6.
- Pasul 4.** Calculează $n=n+2$.
- Pasul 5.** Comunică valoarea lui n . Mergi la Pasul 3.
- Pasul 6.** Terminat.



Clasificarea în funcție de modul de compunere

Datele se clasifică în:

- ✓ **Date simple sau date elementare.** Sunt date independente unele de altele din punct de vedere al reprezentării lor în memorie. Chiar dacă ele pot depinde din punct de vedere logic (valoarea unei date este dependentă de valoarea altei date), ele nu depind din punct de vedere fizic (localizarea unei date pe suportul de memorare nu se face în funcție de locația unei alte date pe suport). În calcularea ariei unui cerc, datele n , i , r și $aria$ sunt date elementare.
- ✓ **Date compuse sau structuri de date.** Sunt colecții de date între care există anumite relații. Fiecare componentă a structurii are o anumită poziție în cadrul structurii, iar toate componentele formează un întreg, astfel încât prelucrarea se poate face atât la nivelul structurii de date (care poate fi considerată o entitate de sine stătătoare), cât și la nivelul fiecărei componente. Pentru fiecare tip de structură de date în limbajul de programare trebuie să fie definiți algoritmi de localizare a componentelor în cadrul structurii de date. Între componentele structurii există și legături de conținut, adică întregul ansamblu de date din colecție poate caracteriza un obiect, o persoană, un fenomen, un proces etc. De exemplu, o colecție cu 12 elemente în care se memorează valorile lunare ale unui contor electric. Structura de date caracterizează în acest caz un proces: consumul lunar de energie electrică. Așadar orice obiect, proces sau fenomen din lumea reală poate fi caracterizat printr-o listă de **proprietăți**. Valorile proprietăților din listă pot fi reprezentate în calculator (lumea virtuală) sub forma unei colecții de date.

Să ne închipuim că într-un algoritm trebuie reprezentată o clasă: profesorul și cei n elevi. Ei reprezintă datele pe care le prelucerează algoritmul. Profesorului îi va corespunde o dată elementară, iar grupului de elevi o structură de date. Locul ocupat în clasă de fiecare dintre ei reprezintă zona de memorie alocată datei: catedra este zona de memorie alocată profesorului, iar grupul de bănci, zona de memorie alocată elevilor. Cele două zone sunt independente. În schimb, în cadrul zonei de bănci (zona de memorie a structurii de date), fiecărui element de structură (elevul) i se alocă un loc într-o bancă, poziția sa putând fi identificată după numărul băncii. Dacă pentru grupul de elevi nu s-ar folosi o structură de date, ci date elementare, fiecărei date elementare va trebui să îi dăm un nume, iar algoritmul ar fi foarte greu de scris. În primul rând nu știm câte date elementare să folosim pentru elevi. Algoritmul trebuie să fie general, deci să funcționeze și pentru o clasă cu 25 de elevi, dar și pentru o clasă cu 30 de elevi. În timpul anului școlar, poate să vină în clasă un elev nou sau poate să plece din clasă un elev. Ce se întâmplă în acest caz cu datele elementare, deoarece la o execuție a algoritmului, atunci când vine un elev nou în clasă ar trebui să apară o nouă dată elementară, iar la o altă execuție a algoritmului, atunci când pleacă un elev din clasă, trebuie să dispară o dată elementară? Soluția este de a folosi o colecție de date. Colecția va avea atâtea elemente câte bănci sunt în clasă. Se va folosi un singur nume de dată care se va atribui colecției, fiecare element identificându-se apoi după numărul băncii.

Studiu de caz

Scop: exemplificarea modului de compunere a datelor.

Enunțul problemei 1: Să se calculeze media aritmetică a n numere introduse de la tastatură.

Enunțul problemei 2: Se introduc n numere de la tastatură. Să se afișeze aceste numere ordonate crescător.

Pentru problema 1 se pot folosi numai date elementare, deoarece, după citirea unui număr prin intermediul datei a , el se prelucrează imediat (se adună la sumă) și variabila de memorie va putea fi refolosită apoi pentru citirea unui alt număr.

Pentru problema 2 nu se poate folosi decât o colecție de date, deoarece, pentru aranjarea într-o anumită ordine a celor n numere citite de la tastatură trebuie să se păstreze în memorie toate aceste date pentru a se putea compara între ele în vederea ordonării.



2.1.2. Tipul datei

Tipul datei determină:

- ✓ dimensiunea zonei de memorie alocate datei (se măsoară în octeți);
- ✓ operatorii care pot fi aplicați pe acea dată;
- ✓ modul în care data este reprezentată în memoria internă (metoda de codificare în binar a valorii datei).

Tipul datei este definit prin dubletul (V, O) , unde:

V = domeniul de definiție intern al datei;

O = mulțimea operatorilor care se pot aplica pe mulțimea de valori ale datei.

Limbajele de programare acceptă următoarele tipuri de date:

- tipul numeric
- tipul logic
- tipul șir de caractere

Tipul numeric

Tipul numeric a fost implementat pentru reprezentarea numerelor întregi sau cu zecimale, pozitive sau negative, și pentru a realiza majoritatea operațiilor matematice întâlnite în practică. Pentru tipul numeric există subtipurile **real** și **întreg**.

Deci: $V = \mathbf{R}$ (mulțimea numerelor reale) sau $\mathbf{I}$ (mulțimea numerelor întregi)

$$O = \mathcal{M}^1 \cup \mathcal{K}^2$$

¹ Mulțimea operatorilor matematici.

² Mulțimea operatorilor relaționali (de comparare).

Constantele de tip numeric se reprezintă prin numere cu semn sau fără semn, folosindu-se punctul pentru separarea părții întregi de partea zecimală: 2; -0.15; 3.175; 20.0.

Tipul logic

Tipul logic sau boolean a fost implementat pentru reprezentarea datelor care nu pot lua decât două valori: adevărat (*true*), pe care o notăm cu *T*, sau fals (*false*), pe care o notăm cu *F*.

Deci: $V = L$ (mulțimea valorilor logice) = $\{T, F\}$
 $O = L^3$

Tipul șir de caractere

Tipul șir de caractere a fost implementat pentru reprezentarea unei mulțimi ordonate de caractere care este tratată ca un tot unitar.

Deci: $V = \{P_C\}$ (mulțimea părților mulțimii C^4)
 $O = R \cup C^5$

În memoria internă, fiecare caracter din șir se reprezintă prin codul său ASCII. Constantele de tip șir de caractere se specifică prin mulțimea ordonată de caractere care compun șirul, delimitată, în funcție de limbajul de programare, de anumite semne speciale: apostrofuri ('Buna ziua') sau ghilimele ("Buna ziua").

alfa	→	identificator de dată elementară
"alfa"	→	construcții greșite
'alfa'		
"alfa"	→	constante de tip șir de caractere
'alfa'		
"500"		
'500'		
500	→	constantă de tip numeric

Constanta de tip numeric 500 este diferită de constanta de tip șir de caractere "500" atât din punct de vedere al modului de reprezentare în memoria internă a calculatorului, cât și din punct de vedere al operatorilor acceptați. De exemplu, asupra constantei numerice se pot aplica operatori matematici și relaționali, iar asupra constantei de tip șir de caractere operatori de concatenare și relaționali. Constanta de tip numeric este reprezentată în memoria internă prin conversia în binar a numărului, iar constanta de tip șir de caractere este reprezentată prin conversia fiecărui caracter din șir în 8 cifre binare corespunzătoare codului ASCII al caracterului respectiv.

³ Mulțimea operatorilor logici.

⁴ Mulțimea caracterelor care este formată din litere, cifre și semne speciale.

⁵ Mulțimea operatorilor de concatenare.

2.2. Operatorii

Operatorii sunt caractere speciale (*, /, >, = etc.) sau cuvinte cheie⁶ (mod, and etc.) prin intermediul cărora se reprezintă operațiile care se efectuează în cadrul unui algoritm. Fiecare limbaj de programare are implementat propriul set de operatori. În acest capitol vor fi prezentați operatorii care se folosesc în cadrul unui algoritm.

Asupra operanzilor dintr-o expresie puteți aplica următorii operatori:

- operatorul de atribuire,
- operatorii matematici,
- operatorul de concatenare a șirurilor de caractere,
- operatorii relaționali,
- operatorii logici.

Operatorii pot fi aplicați numai pe anumite tipuri de operanzi, producând rezultate de un anumit tip. Dacă notăm cu a și b operanzii asupra cărora se aplică operatorul, cu $\odot$ operatorul și cu c rezultatul:

$$a \odot b = c$$

atunci relația între a , b , $\odot$ și c este dată de următorul tabel:

Tipul operanzilor a și b	Tipul operatorului $\odot$	Tipul rezultatului c
numeric	matematic (M)	numeric
numeric	relațional (R)	logic
șir de caractere	concatenare (C)	șir de caractere
șir de caractere	relațional (R)	logic
logic	logic (L)	logic

Operatorii matematici

$$M = \{+, -, *, /, **, \wedge^7, \text{mod}, \text{div}\}$$

Se aplică pe date de tip numeric și furnizează un rezultat de tip numeric.

Operator	Semnificație	Exemplu
+	Adună matematic cei doi operanzi.	$5+2=7$
-	Scade al doilea operand din primul operand.	$7-3=4$
/	Împarte primul operand la al doilea operand.	$7/2=3.5$
*	Înmulțește cei doi operanzi.	$2*4=8$
** ^	Ridică primul operand la puterea furnizată de cel de al doilea operand.	$2**3=8$

⁶ Cuvinte rezervate, care nu mai pot fi folosite ca identificatori pentru date, deoarece ele au un înțeles bine stabilit pentru limbajul de programare.

⁷ Convenție de notare: Semnul | plasat între două elemente are semnificația conjuncției sau, adică pentru operația de ridicare la putere pot fi folosite simbolurile ** sau simbolul ^.

Operator	Semnificație	Exemplu
mod (modulo)	Calculează restul împărțirii primului operand la al doilea operand.	19 mod 4=3
div (împărțire întreagă)	Calculează câtul împărțirii primului operand la al doilea operand.	19 div 4=4

Operatorii relaționali (de comparație)

$$R = \{>, >=, <, <=, =, \#|<>\}$$

Se aplică pe operanzi de tip numeric sau de tip șir de caractere și furnizează un rezultat de tip logic.

Operator	Semnificație	Exemplu
= (egalitate)	Rezultatul este <i>T</i> dacă cei doi operanzi sunt egali.	(5=5)= <i>T</i> (5=7)= <i>F</i>
<> # (diferit)	Rezultatul este <i>T</i> dacă cei doi operanzi sunt diferiți.	(5<>5)= <i>F</i> (5<>7)= <i>T</i>
< (mai mic)	Rezultatul este <i>T</i> dacă primul operand este mai mic decât al doilea operand.	(5<7)= <i>T</i> (7<5)= <i>F</i>
> (mai mare)	Rezultatul este <i>T</i> dacă primul operand este mai mare decât al doilea operand.	(7>5)= <i>T</i> (5>7)= <i>F</i>
<= (mai mic sau egal)	Rezultatul este <i>T</i> dacă primul operand este mai mic sau cel mult egal față de al doilea operand.	(5<=5)= <i>T</i> (7<=5)= <i>F</i>
>= (mai mare sau egal)	Rezultatul este <i>T</i> dacă primul operand este mai mare sau cel mult egal față de al doilea operand.	(7>=5)= <i>T</i> (5>=7)= <i>F</i>

Un operator relațional aplicat pe două șiruri de caractere realizează compararea celor două șiruri de caractere.

Compararea a două caractere este posibilă prin compararea numerică a codurilor ASCII ale celor două caractere. Astfel, codul ASCII al caracterului **d** este 100, iar al caracterului **D** este 68. Deci, caracterul **d** este mai mare decât caracterul **D**. Pentru compararea celor două caractere se va scrie "d">"D" iar în urma executării operației de comparare se va produce rezultatul *T* deoarece operația de comparare se execută între cele două valori numerice (100>68).

Compararea a două șiruri de caractere se face prin compararea codului ASCII al caracterelor din aceeași poziție a fiecărui șir. Dacă cele două șiruri nu au aceeași lungime, șirul cu lungime mai mică este completat la sfârșit, până la egalarea lungimilor, cu caracterul care are codul ASCII 0. Operația de comparare începe cu prima poziție din șir și continuă cu următoarele poziții numai dacă pozițiile anterioare sunt identice în ambele șiruri. De exemplu, șirul de caractere **Idee** este mai mare decât șirul de caractere **IDEE** deoarece în poziția a doua caracterele din cele două șiruri nu mai sunt identice, iar codul ASCII al caracterului **d** este mai mare decât codul ASCII al caracterului **D**. Operația de comparare se oprește după cel de al doilea caracter și nu mai contează codurile caracterelor din pozițiile urmă-

toare. Pentru compararea celor două șiruri de caractere se va scrie "Idee">"IDEE" iar în urma executării operației de comparare se va produce rezultatul *T*.

Operatorul de concatenare

$$\mathcal{C} = \{+\}$$

Se aplică pe operanzi de tip șir de caractere și furnizează un rezultat de tip șir de caractere.

Operator	Semnificație	Exemplu
+ (concatenare)	Reunește două șiruri de caractere într-un singur șir de caractere.	"Buna "+"ziua" = "Buna ziua"

Operatorii logici

$$\mathcal{L} = \{\text{and, or, not}\}$$

Se aplică numai pe operanzi de tip logic și furnizează un rezultat de tip logic.

Operator	Semnificație	<i>a</i>	not <i>a</i>
not (negare)	Schimbă valoarea unui operand cu complementul său: <i>T</i> cu <i>F</i> și <i>F</i> cu <i>T</i> .	<i>T</i>	<i>F</i>
		<i>F</i>	<i>T</i>

Operator	Semnificație	<i>a</i>	<i>b</i>	<i>a</i> and <i>b</i>	<i>a</i> or <i>b</i>
and („și” logic)	Rezultatul este <i>T</i> dacă ambii operanzi au valoarea <i>T</i> , altfel este <i>F</i> .	<i>T</i>	<i>T</i>	<i>T</i>	<i>T</i>
		<i>T</i>	<i>F</i>	<i>F</i>	<i>T</i>
or („sau” logic)	Rezultatul este <i>T</i> dacă cel puțin unul dintre operanzi are valoarea <i>T</i> , altfel este <i>F</i> .	<i>F</i>	<i>T</i>	<i>F</i>	<i>T</i>
		<i>F</i>	<i>F</i>	<i>F</i>	<i>F</i>

Operatorii logici sunt foarte utili atunci când construiți expresii logice care descriu anumite condiții ce vor fi testate, urmând ca în funcție de rezultat să se execute anumite operații. Dacă expresiile logice sunt prea complexe, le puteți simplifica folosind următoarele relații:

$$\text{not } (a \text{ and } b) = (\text{not } a) \text{ or } (\text{not } b)$$

$$\text{not } (a \text{ or } b) = (\text{not } a) \text{ and } (\text{not } b)$$

Operatorul de atribuire

Prin acest operator puteți atribui o anumită valoare unei date:

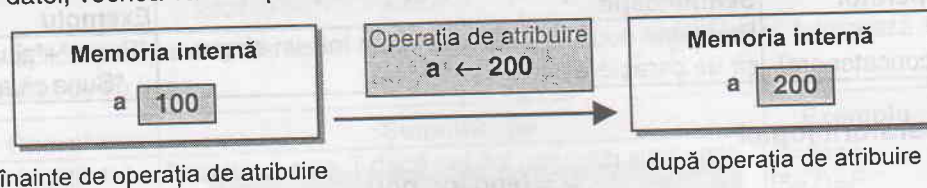
$$\text{nume} \leftarrow \text{expresie}$$

Operația de atribuire se desfășoară în două etape: mai întâi se evaluează expresia după care valoarea obținută se atribuie datei identificată prin *nume*. Rezultatul evaluării expresiei trebuie să fie de același tip cu data identificată prin *nume*.

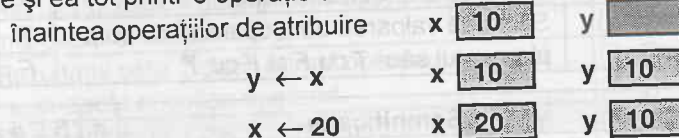
Exemplu	Semnificație
$n \leftarrow 100$	Datei de tip numeric <i>n</i> i se atribuie valoarea 100.

Exemplu	Semnificație
$n \leftarrow n+200$	Se evaluează expresia adunându-se 200 la valoarea datei n . Valoarea obținută în urma evaluării este 300 și se atribuie ca valoare nouă datei n .
$\text{text} \leftarrow \text{"șir de caractere"}$	Datei de tip șir de caractere text i se atribuie valoarea "șir de caractere".

Prin operația de atribuire, în zona de memorie alocată datei, se scrie noua valoare a datei, vechea valoare pierzându-se.



Dacă vrei să păstrezi și vechea valoare, va trebui ca, înainte de operația de atribuire, să o **salvezi prin copiere** într-o altă variabilă de memorie de același tip. Copierea se face și ea tot printr-o operație de atribuire.



Observație: Operația de atribuire se mai folosește în cadrul unui algoritm pentru executarea următoarelor operații:

- ✓ **inițializarea** valorii unor variabile de memorie și
- ✓ **calculul iterativ** al valorii unor variabile de memorie.

Calculul iterativ al valorii unei variabile de memorie înseamnă să-i atribuie acelei variabile de memorie valoarea unei expresii în care unul dintre operanzi este chiar numele acelei variabile de memorie. Tipurile de variabile de memorie cel mai des folosite (din punct de vedere al informației pe care o reprezintă) sunt:

- ✓ **Contorul sau numărătorul.** Este o variabilă de memorie care se folosește pentru a număra anumite cazuri care pot să apară. Înainte de a începe numărătoarea, contorul se inițializează cu valoarea 0 (nu a fost evidențiat încă nici un caz). În timpul procesului de prelucrare, atunci când se întâlnește un caz, valoarea contorului se calculează iterativ prin **incrementare** cu 1 (se crește valoarea contorului cu 1). De exemplu, dacă folosim variabila de memorie **k** pentru contor, **inițializarea** sa se va face cu operația de atribuire: $k \leftarrow 0$, iar **calculul iterativ** cu operația de atribuire: $k \leftarrow k+1$.
- ✓ **Suma.** Este o variabilă de memorie care se folosește pentru calculul iterativ al unei sume (adunarea la valoarea anterioară a unei noi valori). Înainte de a începe calcularea ei, suma se inițializează cu valoarea 0 (elementul neutru pentru operația de adunare, și înseamnă că nu a fost adunată nici o valoare la sumă). De exemplu, dacă folosim variabila de memorie **s** pentru a calcula suma mai

multor valori citite de la tastatură prin intermediul unei variabile de memorie **a**, **inițializarea** sumei se face cu operația de atribuire $s \leftarrow 0$, iar **calculul iterativ** cu operația de atribuire: $s \leftarrow s+a$.

- ✓ **Produsul.** Este o variabilă de memorie care se folosește pentru **calculul iterativ** al unui produs (înmulțirea valorii anterioare cu o nouă valoare). Înainte de a începe calcularea lui, produsul se inițializează cu valoarea 1 (elementul neutru pentru operația de înmulțire, și înseamnă că nu a fost înmulțită nici o valoare cu produsul). De exemplu, dacă folosim variabila de memorie **p** pentru a calcula produsul mai multor valori citite de la tastatură prin intermediul unei variabile de memorie **a**, **inițializarea** produsului se face cu operația de atribuire $p \leftarrow 1$, iar **calculul iterativ** cu operația de atribuire: $p \leftarrow p*a$.

Studiu de caz

Scop. Exemplificarea modului în care folosiți operatorul de atribuire pentru inițializare și calcul iterativ.

Enunțul problemei: Se introduc n numere întregi de la tastatură. Să se numere câte sunt pare și să se calculeze suma și produsul lor.

În urma analizei problemei se obține **specificația programului**:

- ✓ **Funcția programului.** Se numără și se calculează suma și produsul numerelor pare dintr-o mulțime de n numere întregi. Din enunțul problemei rezultă că în procesul de calcul se folosesc numai numere pare. Pentru a evidenția acest caz (aparitia unui număr par în mulțimea de numere citite), se folosește proprietatea numerelor pare: restul împărțirii unui număr par la 2 este 0, care se exprimă prin expresia cu rezultat de tip logic: $(a \bmod 2)=0$. Dacă această expresie are valoarea T , numărul este par.
- ✓ **Informațiile de intrare** sunt numărul de elemente ale mulțimii de numere întregi și numerele citite. Reprezentarea internă a informației se va face prin **datele de intrare**: n pentru numărul de elemente ale mulțimii și a pentru numărul citit curent.
- ✓ Pentru operațiile executate în cadrul algoritmului, se va folosi **data intermediară** i prin care se numără câte numere s-au introdus de la tastatură la un moment dat. Data intermediară i este necesară pentru a afla când se termină procesul de citire a celor n numere, având funcția unui contor. Valoarea inițială a contorului i (înainte de a se citi primul număr) este 0.
- ✓ **Informațiile de ieșire** vor fi numărul de numere pare, suma și produsul lor. Reprezentarea lor internă se va face prin **datele de ieșire**: k pentru numărul de numere (contorul numerelor pare), s pentru suma lor și p pentru produsul lor. Valoarea inițială (înainte de a se citi primul număr) a contorului k și a sumei s este 0, iar a produsului p este 1.

Algoritmul de rezolvare a problemei va fi:

- Pasul 1. Început.
 Pasul 2. Comunică valoarea pentru n .
 Pasul 3. Atribue valoarea inițială pentru contorul numerelor citite i : $i \leftarrow 0$;
 Pasul 4. Atribue valoarea inițială pentru contorul numerelor pare k : $k \leftarrow 0$;
 Pasul 5. Atribue valoarea inițială pentru suma numerelor pare s : $s \leftarrow 0$;
 Pasul 6. Atribue valoarea inițială pentru produsul numerelor pare p : $p \leftarrow 1$;
 Pasul 7. Compară $i \leq n$. Dacă este adevărat, execută Pasul 8; altfel, execută Pasul 14.
 Pasul 8. Comunică valoarea pentru a .
 Pasul 9. Compară $(a \bmod 2) = 0$. Dacă este adevărat execută Pasul 10, altfel execută Pasul 13.
 Pasul 10. Calculează valoarea pentru k : $k \leftarrow k + 1$;
 Pasul 11. Calculează valoarea pentru s : $s \leftarrow s + a$;
 Pasul 12. Calculează valoarea pentru p : $p \leftarrow p * a$;
 Pasul 13. Calculează valoarea pentru i : $i \leftarrow i + 1$. Mergi la Pasul 7.
 Pasul 14. Comunică valorile pentru k , s , p .
 Pasul 15. Terminat.

2.3. Expresiile

Expresia este o combinație validă de operatori și operanzi.

Operanzii pot fi nume de date, constante de tip numeric sau șir de caractere și funcții care în urma evaluării furnizează un singur rezultat. În funcție de tipul operanzilor și al operatorilor, rezultatul expresiei poate fi de tip numeric, șir de caractere sau logic.

Într-o expresie, operatorii se folosesc într-un anumit scop:

- ✓ **Operatorii matematici** – pentru efectuarea calculelor;
- ✓ **Operatorii relaționali și logici** – pentru efectuarea de comparații în vederea luării unor decizii în cadrul algoritmului;
- ✓ **Operatorul de atribuire** – pentru manipularea datelor.

Expresia este validă numai dacă operatorii care leagă operanzii corespund tipului operanzilor pe care îi leagă. De exemplu:

- ✓ Expresia $E \leftarrow 22 + 75$ este o expresie validă deoarece operatorul matematic $+$ leagă doi operanzi exprimați prin două constante de tip numeric.
- ✓ Expresia $E \leftarrow \text{"Ana"} + \text{"-Maria"}$ este o expresie validă deoarece operatorul de concatenare $+$ leagă doi operanzi exprimați prin două constante de tip șir de caractere.
- ✓ Expresia $E \leftarrow 22 + \text{"ani"}$ nu este o expresie validă deoarece operatorul $+$ leagă doi operanzi exprimați prin două constante de tipuri diferite: numeric și șir de caractere. El nu poate fi interpretat nici ca operator matematic, nici ca operator de concatenare.



Dacă într-o expresie nu se poate folosi un operator în care se lucrează

Funcția este o funcție care poate folosi în cadrul unui program care se execută când se evaluează expresia de executării ei. Funcția are un nume și un parametru. Parametrii sunt

De exemplu, funcția care calculează numărul x. Nu este o funcție care evaluează funcția, ci funcția care

Pentru a evalua o expresie pe care le-ați dat

✓ **Precedența** operatorilor.

✓ **Asociațiivitatea** operatorilor același nivel

Fiecare limbaj de program are nivelurile de prioritate într-un algoritm

1. Se evaluează

2. Se evaluează expresiile de tip diferit de prioritate

3. Se evaluează

4. Se evaluează expresiile de prioritate

5. Se evaluează expresiile de prioritate

Asociațiivitatea operatorilor același nivel

De exemplu

✓ $5 * 7 + 140$

$= 140 + 35$

$= 175$

✓ $3 + 3 * 3$

$= 3 + 9$

$= 12$

Informatica

Dacă într-o expresie trebuie să se calculeze radicalul dintr-o dată de tip numeric, nu se poate folosi nici un operator matematic. În acest caz, limbajul de programare în care se lucrează pune la dispoziție o funcție.

Funcția este o prelucrare predefinită de autorii limbajului de programare, care se poate folosi în cadrul unei expresii la fel ca un operand, deoarece ea este de fapt un program care în urma execuției furnizează o valoare chiar prin numele ei. Atunci când se evaluează expresia, funcția este înlocuită cu valoarea returnată în urma executării ei. Funcția se identifică printr-un nume, iar pentru apelarea ei se scrie numele funcției urmat de o listă de parametri precizați între paranteze rotunde. Parametrii sunt valorile cu care se execută funcția la acel apel.

De exemplu, funcția $\sin(x)$ returnează valoarea funcției trigonometrice sinus pentru numărul x . Numele funcției este $\sin$ iar parametrul x este de tip numeric. Dacă se evaluează funcția $\sin(x)$, iar x are valoarea 2.5, în urma execuției programului asociat, funcția va furniza valoarea 0.598.

Pentru a evalua o expresie, calculatorul va executa într-o anumită ordine operațiile pe care le-ai scris. Această ordine este dată de:

- ✓ **Precedența operatorilor.** Este ordinea în care se execută operațiile definite de operatori. Ea determină nivelul de prioritate al operatorilor.
- ✓ **Asociativitatea operatorilor.** Este ordinea în care se evaluează operatorii cu același nivel de prioritate.

Fiecare limbaj de programare are implementată o tabelă de precedență (tabela nivelurilor de prioritate), și pentru fiecare nivel de prioritate o anumită asociativitate. Într-un algoritm vom folosi următoarea tabelă de precedență:

1. Se evaluează funcțiile.
2. Se evaluează operatorii matematici. Operatorii matematici au niveluri de prioritate diferite. Operatorul $**$ are nivel de prioritate 1 fiind primul care se evaluează, operatorii $*$, $/$, div și mod au nivelul de prioritate 2, iar operatorii $+$ și $-$ au nivelul de prioritate 3.
3. Se evaluează operatorii de concatenare.
4. Se evaluează operatorii relaționali. Toți operatorii relaționali au același nivel de prioritate.
5. Se evaluează operatorii logici. Ordinea de prioritate a operatorilor logici este **not**, **and** și **or**, primul fiind cel mai prioritar.

Asociativitatea operatorilor este de la stânga la dreapta, adică toți operatorii care au același nivel de prioritate se evaluează în ordine, de la stânga la dreapta.

De exemplu, următoarele expresii vor fi evaluate astfel:

$$\begin{aligned} \checkmark \quad & 5*7*4 + 4/2 - 2**3\text{mod}2/4 = 5*7*4 + 4/2 - 8\text{mod}2/4 = 35*4 + 4/2 - 8\text{mod}2/4 = \\ & 140 + 4/2 - 8\text{mod}2/4 = 140 + 2 - 8\text{mod}2/4 = 140 + 2 - 0/4 = 140 + 2 - 0 = 142 \end{aligned}$$

$$\checkmark \quad 3<=5 \text{ or } 7>8 = t \text{ or } 7>8 = t \text{ or } f = t$$

$$\checkmark \quad 3<=5 \text{ and not } 7>8 = t \text{ and not } 7>8 = t \text{ and not } f = t \text{ and } t = t$$

Ordinea operațiilor care se vor executa pentru a evalua expresia e :

$$e \leftarrow a*b*d > c \text{ or } a/b=d \text{ and } a+b < d \text{ or not } a > c$$

este:

$e1 \leftarrow b*d$ (operator aritmetic de prioritate 1)	$e7 \leftarrow e4 < d$ (operator relațional)
$e2 \leftarrow a*e1$ (operator aritmetic de prioritate 2)	$e8 \leftarrow a > c$ (operator relațional)
$e3 \leftarrow a/b$ (operator aritmetic de prioritate 2)	$e9 \leftarrow \text{not } e8$ (operator logic)
$e4 \leftarrow a + b$ (operator aritmetic de prioritate 3)	$e10 \leftarrow e6 \text{ and } e7$ (operator logic)
$e5 \leftarrow e2 > c$ (operator relațional)	$e11 \leftarrow e5 \text{ or } e10$ (operator logic)
$e6 \leftarrow e3=d$ (operator relațional)	$e \leftarrow e11 \text{ or } e9$ (operator logic)

Ordinea de evaluare a unei expresii poate fi schimbată prin folosirea parantezelor. Se folosesc numai paranteze rotunde, care pot fi și imbricate.

Atenție! Numărul de paranteze deschise trebuie să fie egal cu numărul de paranteze închise.

De exemplu, următoarea expresie aritmetică prin care se calculează valoarea lui E :

$$E = \frac{a+b}{c} + \frac{b+d}{c+a} + \frac{a+c}{(a+b)^3}$$

va fi scrisă astfel:

$$E \leftarrow (a+b)/c + (b+d)/(c+a) + (a+c)/((a+b)^*3)$$

În acest caz E , a , b , c și d sunt identificatori pentru datele prelucrate de calculator. Data E este o dată de ieșire (rezultatul obținut în urma evaluării expresiei), iar a , b , c și d sunt date de intrare (date care se furnizează calculatorului pentru a putea calcula valoarea lui E).

Ordinea operațiilor care se vor executa pentru a evalua expresia e care conține paranteze și mai multe tipuri de operatori:

$$e \leftarrow ((a/b+c)*4 + a) \bmod b > a*b \text{ or not } (a+c)*b = a*c$$

este:

$e1 \leftarrow a/b$ (operator aritmetic de prioritate 2)
$e2 \leftarrow e1 + c$ (operator aritmetic de prioritate 3)
$e3 \leftarrow e2*4$ (operator aritmetic de prioritate 2)
$e4 \leftarrow e3 + a$ (operator aritmetic de prioritate 3)
$e5 \leftarrow a + c$ (operator aritmetic de prioritate 3)
$e6 \leftarrow e4 \bmod b$ (operator aritmetic de prioritate 2)
$e7 \leftarrow a*b$ (operator aritmetic de prioritate 2)
$e8 \leftarrow e5*b$ (operator aritmetic de prioritate 2)
$e9 \leftarrow a*c$ (operator aritmetic de prioritate 2)
$e10 \leftarrow e6 > e7$ (operator relațional)
$e11 \leftarrow e8=e9$ (operator relațional)
$e12 \leftarrow \text{not } e11$ (operator logic)
$e \leftarrow e10 \text{ or } e12$ (operator logic)

De exemplu, următoarea expresie matematică prin care se calculează valoarea lui e :

$$\frac{\sqrt{a+1} - \sqrt{a-1}}{\sqrt{a+1} + \sqrt{a-1}} \times (2 - a)$$

necesită calculul radicalului de ordin 2 pentru care se folosește funcția **sqrt(x)**. Astfel, atunci când data x are valoarea 9 funcția **sqrt(x)** furnizează valoarea 3, iar când x are valoarea 49 funcția **sqrt(x)** furnizează valoarea 7. Pentru a putea fi evaluată de calculator, expresia va fi scrisă astfel:

$$e \leftarrow ((\text{sqrt}(a+1) - \text{sqrt}(a-1)) / (\text{sqrt}(a+1) + \text{sqrt}(a-1))) * (2 - a)$$

Ordinea operațiilor care se execută pentru a evalua expresia E este:

$e1 \leftarrow a+1$	$e5 \leftarrow \text{sqrt}(e1)$	$e9 \leftarrow e5-e6$
$e2 \leftarrow a-1$	$e6 \leftarrow \text{sqrt}(e2)$	$e10 \leftarrow e7+e8$
$e3 \leftarrow a+1$	$e7 \leftarrow \text{sqrt}(e3)$	$e11 \leftarrow e9/e10$
$e4 \leftarrow a-1$	$e8 \leftarrow \text{sqrt}(e4)$	$e12 \leftarrow 2 - a$
	$e \leftarrow e11 * e12$	

Precondițiile unei expresii

Expresia definește cum trebuie să prelucreze calculatorul anumite valori ale datelor. De fiecare dată când calculatorul evaluează expresia, sunt prelucrate valorile curente ale datelor variabile care apar ca operanzi în cadrul expresiei. În unele cazuri trebuie stabilite anumite condiții pentru valorile acestor date pentru a se putea evalua expresia, adică trebuie stabilite **precondițiile expresiei**.

Precondițiile expresiei reprezintă un ansamblu de restricții și constrângeri impuse datelor care apar în expresie ca operanzi.

Aceasta înseamnă că, dacă notăm precondiția cu P și expresia cu E , atunci:

- ✓ Dacă precondiția P este adevărată (are valoarea *true*), expresia poate fi calculată cu acele valori ale datelor.
- ✓ Dacă precondiția P este falsă (are valoarea *false*), expresia nu poate fi calculată cu acele valori ale datelor, deoarece fie se obține o valoare eronată, fie se poate provoca o eroare de execuție a programului.

Precondițiile cele mai des întâlnite sunt:

- ✓ **Constrângerile operației de împărțire.** În algoritmi puteți folosi trei operatori pentru împărțire: $/$, div și mod . Pentru toate aceste operații există **precondiția ca împărțitorul să fie diferit de zero**. De exemplu, pentru expresia $a/(c+d)$, precondiția este $c+d \neq 0$.
- ✓ **Constrângerile argumentului unei funcții.** Argumentele unor funcții trebuie să îndeplinească o anumită condiție. De exemplu, funcția **sqrt(x)** nu se poate aplica pe o valoare negativă (nu se poate extrage radicalul pătrat dintr-o valoare negativă). De exemplu, pentru expresia $-b + \text{sqrt}(b*b - 4*a*c)$ precondiția este $b*b - 4*a*c \geq 0$, iar pentru expresia $(-b + \text{sqrt}(b*b - 4*a*c)) / (2*a)$ precondiția este $(a > 0)$ and $(b*b - 4*a*c \geq 0)$.

- ✓ **Constrângerile pentru ca formula să fie validă.** Formula folosită pentru a calcula o valoare poate fi validă numai pentru anumite valori ale datelor care sunt folosite ca operanzi. De exemplu, operatorii **div** și **mod** se pot aplica numai pe date de tip întreg.

Evaluare

Răspundeți:

1. Ce este identificatorul unei date? Dați cinci exemple de identificatori.
2. Ce este o variabilă de memorie?
3. Prin ce se deosebesc datele de intrare, datele de ieșire și datele de manevră? Dați un exemplu de problemă în care să folosiți toate aceste tipuri de date. Precizați pentru fiecare dată de ce tip este.
4. Ce este tipul datei? Cum influențează tipul datei o variabilă de memorie?
5. Câte tipuri de date există? Ce operatori puteți folosi pentru fiecare tip de dată?
6. Ce este precedența operatorilor? Dar asociativitatea operatorilor?

Rezolvați:

1. Se consideră următoarea problemă:

Se dau două numere întregi **a** și **b**. În funcție de răspunsul la un mesaj întrebare (de exemplu "Ce operație doriți?") se va calcula: dacă răspunsul este litera **x**, media aritmetică a celor două numere; dacă răspunsul este litera **y**, media geometrică a celor două numere; dacă răspunsul este litera **z**, câtul și restul împărțirii numărului **a** la numărul **b**; dacă răspunsul este orice altă literă se va afișa un mesaj de informare (de exemplu "Alegere greșită").

Pentru rezolvarea acestei probleme cu ajutorul unui program de calculator, se vor folosi mai multe date elementare, care să permită generalizarea problemei. Completați următorul tabel prin care veți face o analiză a datelor folosite:

Identificatorul datei	Reprezintă	Tipul datei (de intrare, de ieșire, ...)	Tipul datei (numeric, logic, ...)	Observații (constantă, formula de calcul, ...)

2. Dacă într-un algoritm există variabilele **a** de tip caracter, **b** de tip întreg și **c** de tip logic și se atribuie acestor date următoarele valori inițiale:

a: '4' **b:** 8 **c:** false

evaluați următoarele expresii:

Expresia	Rezultat	Expresia	Rezultat
(b>15) or c		not c or (a='a')	
a>='0' and 'a<='9'		(a > b) and c	

3. Pentru următoarele valori ale datelor:

i 4 **j** 6 **k** 8 **l** 1 **m** 10

Informatica

evaluați următoarele expresii:

Expresia	Rezultat	Expresia	Rezultat
$i+j \cdot l-m$		$i-j+k-l+m$	
$(i+j) \text{ div } (m-j)$		$3^j \text{ mod } m-i$	
$(j+(m-l \cdot (j+i))-k)+5$		j^j-i-k	

4. Legați prin linii fiecare element din coloana **Construcția** de elementul corespunzător din coloana **Reprezintă**:

Construcția:

alfa	1
"alfa"	2
'10'''	3
'20'''	4
'alfa'	5
5000	6
"alfa"	7
'500'	8
"120"	9

Reprezintă:

- a identificator dată elementară
b constantă de tip șir de caractere
c constantă de tip numeric
d construcție greșită

5. Se dau următoarele secvențe de operații de atribuire:

(S1): $a \leftarrow 1; b \leftarrow 2; c \leftarrow 3; d \leftarrow 4$
(S2): $a \leftarrow b; b \leftarrow c; c \leftarrow d; d \leftarrow a$
(S3): $b \leftarrow c; c \leftarrow d; d \leftarrow a; a \leftarrow b$
(S4): $c \leftarrow d; d \leftarrow a; a \leftarrow b; b \leftarrow c$

Să se precizeze valorile variabilelor de memorie a, b, c și d obținute în urma executării operațiilor de atribuire, în ordine:

- a) (S1); (S2); (S3)
b) (S1); (S3); (S2)
c) (S1); (S2); (S2); (S3)
d) (S1); (S2); (S3); (S4)

Ce concluzii puteți trage în urma executării operațiilor de la punctele a) și b)?

6. Scrieți următoarele expresii matematice în forma acceptată de calculator:

$$E_1 = \frac{x^3 - 1}{x(x-3)(x-1)}$$

$$E_2 = \frac{a^2 + (a+b)^3}{ab^2} + a^2b$$

$$E_3 = \frac{4\left(\frac{a}{bc} + c\right) + (ab)^2 + 3\frac{a}{bc}}{2a^3}$$

7. Descrieți ordinea de evaluare a celor trei expresii de la problema 6.
8. Scrieți următoarea expresie matematică în forma acceptată de calculator folosind pentru calculul radicalului de ordin 2 din x funcția **sqrt(x)**:

Informatica

16. În unele aplicații este necesar să se precizeze că o dată poate lua numai anumite valori. Aceste descrieri se folosesc pentru a verifica dacă valoarea datei obținută în urma unor operații de prelucrare aparține mulțimii de valori permise. De exemplu, considerăm mulțimea $A = (0, 10] \cup \{20\}$ și o dată de tip numeric x . Pentru a verifica dacă $x \in A$, se construiește o expresie logică $c1$ care descrie condiția de apartenență a datei x la mulțimea A , urmând ca prin testarea valorii acestei expresii să se verifice dacă valoarea datei x este corectă: dacă expresia $c1$ are valoarea *adevărat*, $x \in A$, iar dacă expresia $c1$ are valoarea *fals*, $x \notin A$. Pentru acest exemplu se construiește expresia $c1$:

$$c1 \leftarrow (x > 0 \text{ and } x \leq 10) \text{ or } x = 20$$

Dacă vrei să precizezi că valoarea datei x nu trebuie să aparțină mulțimii precizate mai sus, se va construi expresia $c2$ prin negarea expresiei $c1$:

$$c2 \leftarrow \text{not } e = \text{not}((x > 0 \text{ and } x \leq 10) \text{ or } x = 20) = \text{not}(x > 0 \text{ and } x \leq 10) \text{ and } \text{not}(x = 20) = \\ = (\text{not } x > 0 \text{ or } \text{not } x \leq 10) \text{ and } x \neq 20 = (x \leq 0 \text{ or } x > 10) \text{ and } x \neq 20$$

Observați că, în urma negării, un operator relațional se înlocuiește cu complementarul său: $<>$ cu $=$, $>$ cu $<$, $\leq$ cu $>$ și invers.

Pornind de la acest exemplu, să considerăm mulțimea $A = [-10, 10] \cup (20, 30) \cup \{50\}$ și o dată de tip numeric x . Precizați expresia $c1$ prin care se verifică dacă $x \in A$ și expresia $c2$ prin care se verifică dacă $x \notin A$.

17. Fie mulțimea $A = [-30, -20] \cup (-15, 10] \cup (40, 50) \cup \{-50, 100\}$ și o dată de tip numeric x . Precizați expresia $c1$ prin care se verifică dacă $x \in A$ și expresia $c2$ prin care se verifică dacă $x \notin A$.
18. De exemplu, dacă datele a , b și c reprezintă lungimile laturilor unui triunghi ABC, pentru a verifica dacă acest triunghi este un triunghi echilateral (adică $a=b=c$) trebuie să se evalueze expresia logică $c1$:

$$c1 \leftarrow (a = b) \text{ and } (b = c)$$

Dacă expresia are valoarea *adevărat*, triunghiul este echilateral. Această expresie descrie condiția pe care trebuie să o îndeplinească cele trei date a , b și c ca să reprezinte laturile unui triunghi echilateral. Pornind de la acest exemplu, precizați condiția pe care trebuie să o îndeplinească cele trei date pentru ca triunghiul să fie:

- a) isoscel, b) dreptunghic, c) dreptunghic isoscel.

19. Precizați **condiția** pe care trebuie să o îndeplinească trei date ca să reprezinte lungimile laturilor unui triunghi (suma lungimilor a două laturi trebuie să fie mai mare decât lungimea celei de a treia laturi, oricare ar fi cele două laturi).
20. Scrieți **condiția** $c1$ prin care testați dacă patru date de tip numeric a , b , c și d pot reprezenta laturile unui paralelogram și condiția $c2$ prin care testați dacă aceste date nu pot reprezenta laturile unui paralelogram.
21. Scrieți **condiția** prin care testați dacă valoarea unui număr întreg n este:
- un număr impar,
 - un număr divizibil cu 3 sau cu 5,
 - un număr divizibil cu 3 și cu 5,
 - un număr divizibil cu 3 dar nu și cu 5,

e) un pătrat perfect (veți folosi funcțiile: $\text{int}(x)$ – pentru partea întreagă din x și $\text{sqrt}(x)$ – pentru radical de ordinul 2 din x).

22. Pentru a testa ultima cifră a unui număr întreg n , aceasta se extrage cu expresia $\text{cifra} \leftarrow n \bmod 10$ (restul împărțirii numărului la 10). De exemplu, pentru a testa dacă ultima cifră a unui număr n este 2 se folosește condiția $c \leftarrow n \bmod 10 = 2$. Scrieți condiția prin care testați dacă ultima cifră a unui număr n este:

- 3 sau 5,
- diferită de 3 și de 5,
- pară,
- impară,
- multiplu de 3,
- multiplu de 3 sau de 5.

23. În data *alfa* se memorează un caracter. Pentru a afla dacă acest caracter este litera *a* sau *A*, se va testa expresia *c1* care descrie această condiție:

$c1 \leftarrow (\text{alfa} = "a") \text{ or } (\text{alfa} = "A")$

iar pentru a afla dacă acest caracter este una dintre literele *a*, *b*, *c*, *d*, *A*, *B*, *C* sau *D*, se va testa expresia *c2* care descrie această condiție:

$c2 \leftarrow (\text{alfa} \geq "a" \text{ and } \text{alfa} \leq "d") \text{ or } (\text{alfa} \geq "A" \text{ and } \text{alfa} \leq "D")$

Pornind de la acest exemplu precizați condiția prin care se poate testa caracterul memorat în data *alfa*, astfel încât:

- să nu fie o cifră (de exemplu "1" sau "8"),
- să fie o cifră,
- să fie o vocală,
- să nu fie o vocală,
- să fie o consoană (testul se va referi numai la literele mici),
- să nu fie o consoană (testul se va referi numai la literele mici),
- să aparțină mulțimii de caractere $C = \{a, x, y, z, w, B, X, Y, Z\}$,
- să nu aparțină mulțimii de caractere $C = \{a, x, y, z, w, B, X, Y, Z\}$,
- să aparțină mulțimii de caractere $C = \{a, l, m, n, o, p, C, D, F\}$,
- să nu aparțină mulțimii de caractere $C = \{a, l, m, n, o, p, C, D, F\}$,
- să fie spațiu, punct sau virgulă,
- să nu fie spațiu, punct sau virgulă.

24. Se consideră următorul enunț: *Se citesc n numere naturale. Să se calculeze suma și produsul celor divizibile cu 3 sau cu 5. Analizați problema și scrieți algoritmul.*

25. Se consideră următorul enunț: *Se citesc n numere naturale. Să se numere câte sunt divizibile cu 2, cu 3 și cu 6. Analizați problema și scrieți algoritmul.*

26. Se consideră următorul enunț: *Se citesc mai multe numere întregi până la întâlnirea numărului 0. Să se calculeze media aritmetică a numerelor pozitive. Analizați problema. Identificați cazul în care trebuie să folosiți precondiția expresiei. Scrieți algoritmul.*

27. Se consideră următorul enunț: *Se citesc mai multe numere naturale până la întâlnirea numărului 0. Să se calculeze media aritmetică a numerelor care au*

Informatica

ultima cifră 5. Analizați problema. Identificați cazul în care trebuie să folosiți pre-condiția expresiei. Scrieți algoritmul.

Alegeți:

1. Datei a i se atribuie valoarea $"2+1="$. Dacă se afișează conținutul ei pe ecran, veți vedea:
a) "3" b) 3 c) "2+1=" d) 2+1=
2. Datei a i se atribuie expresia $"25"+"75"$. Valoarea datei va fi:
a) "2575" b) 100 c) "100"
3. Prin negarea expresiei $n \bmod 2 < 0 \text{ OR } n > 10$ se testează dacă data n este:
a) un număr par mai mare decât 10;
b) un număr impar mai mare decât 10;
c) un număr par mai mic sau egal cu 10;
d) un număr impar mai mic sau egal cu 10;
4. Care dintre expresiile următoare testează dacă n este un număr natural multiplu de 3 sau de 5:
a) $n > 0$ and $n \bmod 3 = 0$ or $n \bmod 5 = 0$
b) $n > 0$ and $n \bmod 3 = 0$ and $n \bmod 5 = 0$
c) $n > 0$ and $(n \bmod 3 = 0 \text{ or } n \bmod 5 = 0)$
d) $n > 0$ or $n \bmod 3 = 0$ or $n \bmod 5 = 0$
5. Care dintre expresiile următoare testează dacă n este un număr natural care nu se divide prin 3 și prin 5:
a) $n > 0$ and $n \bmod 3 < 0$ or $n \bmod 5 < 0$
b) $n > 0$ and $n \bmod 3 < 0$ and $n \bmod 5 < 0$
c) $n > 0$ and $(n \bmod 3 < 0 \text{ or } n \bmod 5 < 0)$
d) $n > 0$ or $n \bmod 3 < 0$ or $n \bmod 5 < 0$
6. Care dintre expresiile următoare testează dacă n este un număr natural care are ultima cifră diferită de 5 și de 0:
a) $n > 0$ and $n \bmod 10 < 5$ or $n \bmod 10 < 0$
b) $n > 0$ and $n \bmod 10 < 5$ and $n \bmod 10 < 0$
c) $n > 0$ and $(n \bmod 10 < 5 \text{ or } n \bmod 10 < 0)$
d) $n > 0$ or $n \bmod 10 < 5$ or $n \bmod 10 < 0$
7. Care dintre expresiile următoare testează dacă n este un număr natural care are ultima cifră 0, 2, 4 sau 8:
a) $n > 0$ and $n \bmod 10 \bmod 2 = 0$ or $n \bmod 10 \bmod 3 < 0$
b) $n > 0$ and $n \bmod 10 \bmod 2 = 0$ and $n \bmod 10 \bmod 3 < 0$
c) $n > 0$ and $(n \bmod 10) \bmod 2 = 0$ or $(n \bmod 10) \bmod 3 < 0$
d) $n > 0$ and $((n \bmod 10) \bmod 2 = 0 \text{ or } (n \bmod 10) \bmod 3 < 0)$
8. În urma cărei operații de atribuire, data b va avea valoarea ultimei cifre a numărului întreg a :
a) $b \leftarrow a \text{ div } 10$ c) $b \leftarrow a - a \text{ div } 10 * 10$
b) $b \leftarrow a - a \text{ div } 10$ d) $b \leftarrow a - a \bmod 10$

3. Algoritmii

3.1. Reprezentarea algoritmilor

Algoritmul este un concept abstract. **Reprezentarea algoritmului înseamnă implementarea fizică a algoritmului.** Chiar dacă algoritmul este unic, el poate avea mai multe reprezentări fizice. De exemplu, algoritmul de rezolvare a ecuației de gradul întâi poate fi reprezentat prin calculele efectuate pe hârtie de fiecare dată când se rezolvă manual o ecuație de gradul întâi, prin circuite electronice, dacă s-ar construi o mașină electronică numai pentru rezolvarea ecuației de gradul întâi sau prin instrucțiunile unui program care descriu pentru calculator algoritmul de rezolvare.

Când construiești un algoritm trebuie să ții cont de următoarele reguli:

- ✓ să definești exact datele asupra cărora lucrează algoritmul (datele de intrare, datele de ieșire și datele intermediare);
- ✓ să definești exact operațiile care se vor executa cu datele asupra cărora lucrează algoritmul;
- ✓ să definești exact noțiunea de structură de control a algoritmului;
- ✓ să definești exact succesiunea de structuri care formează algoritmul.

Algoritmul prin care se descrie o problemă care trebuie să fie rezolvată de calculator nu trebuie să fie ambiguu deoarece, în cazul exprimărilor neclare, calculatorul nu poate să opteze singur pentru o anumită posibilitate. Pentru a evita ambiguitatea descrierii unui algoritm printr-un limbaj natural (limba în care vorbim) se poate folosi pentru reprezentarea lui un limbaj artificial numit **pseudocod**, apropiat de limbajul de programare, dar care este puțin formalizat și nu este constrâns de regulile de sintaxă ale limbajului de programare (de exemplu, în pseudocod se poate folosi exprimarea **dacă... atunci... altfel** – în limba română – sau formularea **if... then... else** – în limba engleză – care sunt foarte apropiate de limbajul natural, dar care permit descrierea unor operații specifice din algoritm).

Pseudocodul (codul fals) este considerat un cod fals deoarece nu poate fi folosit pentru a exprima instrucțiunile care se dau calculatorului pentru a rezolva problema descrisă de algoritm (nu poate fi folosit ca limbaj de programare). El folosește expresii din limbajul natural în care exprimarea acțiunilor care se execută se face prin propoziții care se termină prin simbolul punct și virgulă (;). În propoziții se folosesc **cuvinte cheie** pentru descrierea structurilor de control și a operațiilor de comunicare. O propoziție care reprezintă un pas de comunicare sau de acțiune începe obligatoriu cu un verb.

Pseudocodul permite și descrierea datelor asupra cărora acționează algoritmul. Pentru precizarea tipului de dată se folosesc cuvinte cheie. De exemplu, se pot folosi următoarele cuvinte cheie: **întreg** – tipul numeric întreg; **real** – tipul numeric real; **logic** – tipul logic; **caracter** – tipul caracter; și **șir** – tipul șir caracter. Cuvântul

cheie care precizează tipul este urmat de o listă prin care se enumeră identificatorii datelor care corespund aceluiași tip.

În cazul pasului de comunicare, verbul este **citește** (read) pentru o operație de intrare și **scrie** (write) pentru o operație de ieșire. Verbul este urmat de lista datelor care se comunică. Lista conține date variabile reprezentate prin identificatorii lor. În cazul unei operații de scriere lista poate conține și date constante, reprezentate prin valoare (de exemplu, un mesaj reprezentat printr-o constantă de tip șir de caractere). Elementele listei se separă prin virgulă.

Cuvintele cheie pot fi în limba română (pseudocodul în limba română) sau în limba engleză (pseudocodul în limba engleză). Pseudocodul în limba engleză este mai aproape de cuvintele cheie folosite în instrucțiunile unui limbaj de programare. Din această cauză, vor fi prezentate ambele versiuni de pseudocod, urmând ca în aplicații să folosim numai pseudocodul în limba română.

Pentru a delimita secvența de descriere a datelor de secvența de descriere a pașilor algoritmului, pașii algoritmului vor fi încadrați de cuvintele cheie **început...** și **sfârșit**. (begin... end.).

De exemplu, pseudocodul pentru descrierea algoritmului de rezolvare a ecuației de gradul întâi este:

```
real a,b,z;  
început  
  citește a,b;  
  dacă a=0  
    atunci  
      dacă b=0  
        atunci scrie "Ecuația are o infinitate de soluții";  
      altfel scrie "Ecuația nu are soluții";  
    sfârșit_dacă;  
  altfel z ← -b/a;  
    scrie "Soluția ecuației este ", z;  
  sfârșit_dacă;  
sfârșit.
```

Observații:

1. În pseudocod operația de comparație a fost exprimată prin cuvintele cheie **dacă** și **sfârșit_dacă**. Aceste două cuvinte formează o **structură de control**, adică o entitate din cadrul algoritmului prin care se descrie modul în care pașii algoritmului își predau controlul unul altuia.
2. În pseudocod, pentru orice structură de control se folosește o **pereche de cuvinte cheie**: primul cuvânt precizează începutul structurii (cuvântul **dacă**), iar al doilea cuvânt precizează sfârșitul structurii (cuvântul **sfârșit_dacă**).
3. Ca să fiți siguri că ați scris corect structurile de control, verificați ca numărul de structuri deschise să fie egal cu numărul de structuri închise (de exemplu, numărul de cuvinte **dacă** să fie egal cu numărul de cuvinte **sfârșit_dacă**).

4. Pentru a urmări mai ușor dacă structurile sunt închise corect, scrieți indentat corpul structurii față de cuvintele cheie cu care începeți și cu care terminați structura. Scrieți la același nivel de indentare propozițiile care se execută secvențial. De exemplu, în structura de control descrisă în pseudocod prin cuvintele cheie **dacă... sfârșit_dacă**, scrieți indentat față de cuvintele cheie **atunci** și **altfel** pașii care se execută pentru cele două valori ale condiției testate.

3.2. Principiile programării structurate

Algoritmul este format din pașii care urmează să se execute și ordinea în care se vor executa pentru a rezolva problema. Algoritmul pentru rezolvarea ecuației de gradul întâi conține nouă pași (vezi algoritmul de la pagina 9), iar ordinea de executare depinde de valoarea celor doi coeficienți **a** și **b**:

- ✓ Dacă **a** = 0 și **b** = 0, se execută în ordine pașii: **1, 2, 3, 4, 5, 9.**
- ✓ Dacă **a** = 0 și **b** ≠ 0, se execută în ordine pașii: **1, 2, 3, 4, 6, 9.**
- ✓ Dacă **a** ≠ 0, se execută în ordine pașii: **1, 2, 3, 7, 8, 9.**

Structura de control a algoritmului definește **ordinea de executare a pașilor**, adică ordinea în care un pas predă controlul altui pas și prin care se determină fluxul controlului. În cadrul algoritmilor pot fi folosite trei tipuri de structuri de control:

- structura liniară,
- structura alternativă,
- structura repetitivă.

3.2.1. Structura liniară

Structura liniară sau secvențială este structura în care pașii se execută în ordinea în care au fost scriși: **Pasul 1, Pasul 2, ..., Pasul i, Pasul i+1, ..., Pasul n-1, Pasul n.** Fiecare pas predă controlul pasului următor (**Pasul i** predă controlul **Pasului i+1**). Un pas al structurii secvențiale se execută numai dacă au fost executați toți pașii care îl preced. Structura secvențială nu folosește decât pași de tip acțiune și comunicare.

Exemplu

Se introduc de la tastatură trei numere **a, b, c**. Să se calculeze media aritmetică dintre **a** și **b** și media aritmetică dintre **b** și **c**. Pentru calcularea celor două medii se vor folosi două variabile de memorie **m1** și **m2** și se vor executa în ordine următoarele acțiuni:

- Pasul 1.** Început.
Pasul 2. Comunică valorile pentru **a, b** și **c**.
Pasul 3. Calculează $m1 \leftarrow (a+b)/2$.
Pasul 4. Calculează $m2 \leftarrow (b+c)/2$.
Pasul 5. Comunică valorile lui **m1** și **m2**.
Pasul 6. Terminat.

```

întreg a,b,c;
real m1,m2;
început
  citește a,b,c;
  m1 ← (a+b)/2;
  m2 ← (b+c)/2;
  scrie m1,m2;
sfârșit.

```

Informatică

Ordinea în care se execută cei șase pași este ordinea prezentată: **1, 2, 3, 4, 5, 6**, oricare ar fi valorile pentru **a, b, c**. **Pasul 1** predă controlul **Pasului 2**, **Pasul 2** predă controlul **Pasului 3**, **Pasul 3** predă controlul **Pasului 4**, **Pasul 4** predă controlul **Pasului 5** și **Pasul 5** predă controlul **Pasului 6**.

Algoritmul a fost transpus în pseudocod.

Probleme care se pot rezolva cu ajutorul structurii secvențiale:

1. Se citesc trei numere întregi de la tastatură. Să se calculeze media aritmetică.
2. Se citesc dimensiunile pentru laturile unui triunghi. Să se calculeze aria și perimetrul triunghiului.

3.2.2. Structura alternativă

Prin această structură, se face selectarea între două sau mai multe acțiuni, în funcție de anumite condiții.

Există două tipuri de structuri alternative:

- structura alternativă simplă,
- structura alternativă generalizată.

Structura alternativă simplă

La acest tip de structură se face selectarea între două acțiuni, în funcție de îndeplinirea sau neîndeplinirea unei condiții. De exemplu:

- ✓ Dacă anul este bisect, atunci împarte totalul la 366; altfel, împarte la 365.
Condiția este tipul anului: bisect sau nu.
- ✓ Dacă vânzările au scăzut, reduceți prețul cu 5%. Condiția este scăderea vânzărilor.
- ✓ Dacă unghiul are 90° , atunci este un unghi drept. Condiția este valoarea unghiului.
- ✓ Dacă triunghiul are toate laturile egale, atunci este echilateral. Condiția este relația de egalitate între laturile triunghiului.
- ✓ Dacă patrulaterul are toate laturile egale și un unghi de 90° , atunci este un pătrat. Condiția este relația de egalitate între laturile patrulaterului și valoarea unui unghi al patrulaterului.
- ✓ Dacă ai sub 18 ani, atunci ești minor sau, dacă ai peste 18 ani, atunci ești major. Condiția este vârsta de 18 ani.
- ✓ Dacă vei fi în oraș la ora 13, luăm prânzul împreună. Condiția este prezența în oraș la ora 13.
- ✓ Dacă nu ai 10.000.000 de lei, împrumută-te la bancă. Condiția este suma de 10.000.000 lei.
- ✓ Dacă valoarea polinomului este 0 pentru $x=2$, polinomul se divide prin $x-2$. Condiția este valoarea 0 a polinomului pentru $x=2$.

Exemplu

Se introduce de la tastatură un număr n . Să se calculeze inversul acestui număr, **inv**, definit astfel:

$$\text{inv} = \begin{cases} 1/n & \text{pentru } n \neq 0 \\ 0 & \text{pentru } n = 0 \end{cases} \quad \text{Altfel spus: dacă } n \text{ este diferit de } 0, \text{ inversul are valoarea } 1/n, \text{ altfel are valoarea } 0.$$

Pentru calculul inversului se vor executa în ordine următoarele acțiuni:

Pasul 1. Început.

Pasul 2. Comunică valoarea pentru n .

Pasul 3. Dacă $n \neq 0$, atunci execută **Pasul 4**, altfel execută **Pasul 5**.

Pasul 4. Calculează $\text{inv} \leftarrow 1/n$.

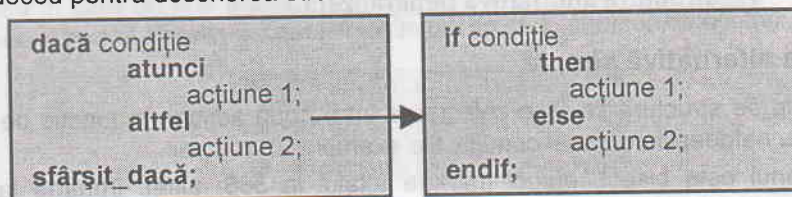
Pasul 5. Calculează $\text{inv} \leftarrow 0$.

Pasul 6. Comunică valoarea lui inv .

Pasul 7. Terminat.

Executarea uneia dintre cele două acțiuni posibile depinde de condiția precizată printr-o expresie logică. Dacă expresia logică are valoarea *adevărat*, se execută *acțiune 1*, iar dacă expresia logică are valoarea *fals*, se execută *acțiune 2*. Cele două acțiuni pot fi descrise printr-un singur pas sau prin mai mulți pași.

Observați că acest tip de structură poate fi descris prin raționamentul **dacă... atunci... altfel... sfârșit_dacă** (**if... then... else... endif**) pe care îl putem folosi în pseudocod pentru descrierea structurii.



Cuvântul **dacă** (**if**) marchează începutul structurii, iar cuvântul **sfârșit_dacă** (**endif**) sfârșitul structurii. Cuvintele **dacă** (**if**) și **atunci** (**then**) delimitează expresia logică ce se evaluează, cuvântul **atunci** (**then**) marchează începutul secvenței de pași care descriu acțiunea care se va executa dacă expresia logică are valoarea *adevărat*, și cuvântul **altfel** (**else**) marchează începutul secvenței de pași care descriu acțiunea care se va executa dacă expresia logică are valoarea *fals*.

Observați că se execută secvențial **Pasul 1**, **Pasul 2**, **Pasul 3**, după care ordinea secvențială este abandonată și se execută **Pasul 4** sau **Pasul 5**, în funcție de valoarea expresiei logice $n \neq 0$, după care se reia ordinea secvențială, executându-se **Pasul 6** și **Pasul 7**.

```

întreg n;
real inv;
început
    citește n;
    dacă n <> 0
        atunci
            inv ← 1/n;
        altfel
            inv ← 0;
    sfârșit_dacă;
    scrie inv;
sfârșit.

```

Exemplul a fost transpus în pseudocod.

Un caz particular de structură alternativă este **structura alternativă cu o ramură vidă**, în care *acțiune 2* nu conține nici un pas. Deoarece pentru valoarea *fals* a expresiei logice nu se execută nici o acțiune, din pseudocod va fi eliminat cuvântul

altfel (else) care marchează începutul secvenței de pași ce descriu acțiunea care se va executa dacă expresia logică are valoarea fals.

Exemplu

Se introduce de la tastatură un număr n . Să se înlocuiască numărul n cu modulul său, definit astfel:

$$\text{mod}(n) = \begin{cases} n & \text{pentru } n \geq 0 \\ -n & \text{pentru } n < 0 \end{cases}$$

Se vor executa în ordine următoarele acțiuni:

- Pasul 1.** Început.
- Pasul 2.** Comunică valoarea pentru n .
- Pasul 3.** Dacă $n < 0$, atunci execută **Pasul 4**, altfel execută **Pasul 5**.
- Pasul 4.** Calculează $n \leftarrow -n$.
- Pasul 5.** Comunică valoarea lui n .
- Pasul 6.** Terminat.

```

întreg n;
început
citește n;
dacă n < 0
    atunci
        n ← -n;
sfârșit dacă;
scrie n;
sfârșit.
    
```

Exemplul a fost transpus în pseudocod.

Observație: În cazul structurii alternative cu o ramură vidă, acțiunea se va executa pe ramura pentru care condiția are valoarea adevărat. Dacă, după ce ai gândit algoritmul, acțiunea se execută pe ramura pentru care condiția are valoarea fals, îl veți corecta prin negarea condiției.

Structura alternativă generalizată

La acest tip de structură se face selectarea între mai multe acțiuni, în funcție de o variabilă de memorie numită **selector**, care poate lua mai multe valori, dintr-o mulțime ordonată de elemente de același tip cu selectorul. De exemplu:

- ✓ Dacă simbolul dintre doi operanzi este +, adună operanzii, dacă este minus, scade din primul operand al doilea operand, dacă este *, înmulțește operanzii, dacă este /, împarte operanzii, iar dacă este alt operand, expresia este eronată. În acest caz selectorul este simbolul, iar mulțimea de valori este formată din simbolurile: +, -, * și /.
- ✓ Dacă este clasa a IX-a A, sunt 27 de elevi, dacă este clasa a IX-a B, sunt 28 de elevi, iar dacă este clasa a IX-a C, sunt 28 de elevi. În acest caz selectorul este numele clasei, iar mulțimea de valori este formată din numele de clase: a IX-a A, a IX-a B și a IX-a C.

Exemplu

Se introduc de la tastatură numerele întregi n , a , b și c . Să se calculeze valoarea lui e , definit astfel:

$$e = \begin{cases} (a+b)/c & \text{pentru } n=1 \\ (b+c)/a & \text{pentru } n=2 \\ (c+a)/b & \text{pentru } n=3 \end{cases}$$

Pentru calculul expresiei e , se vor executa în ordine următoarele acțiuni:

- Pasul 1.** Început.
- Pasul 2.** Comunică valorile pentru n , a , b , c .

- Pasul 3. Dacă $n=1$, atunci execută **Pasul 4**, altfel execută **Pasul 5**.
 Pasul 4. Calculează $e \leftarrow (a+b)/c$. Treci la **Pasul 9**.
 Pasul 5. Dacă $n=2$, atunci execută **Pasul 6**, altfel execută **Pasul 7**.
 Pasul 6. Calculează $e \leftarrow (b+c)/a$. Treci la **Pasul 9**.
 Pasul 7. Dacă $n=3$, atunci execută **Pasul 4**, altfel execută **Pasul 9**.
 Pasul 8. Calculează $e \leftarrow (c+a)/b$. Treci la **Pasul 9**.
 Pasul 9. Comunică valoarea lui e .
 Pasul 10. Terminat.

Executarea uneia dintre acțiunile posibile depinde de valoarea selectorului s . Dacă selectorul are valoarea $v1$, se execută acțiune 1, dacă are valoarea $v2$, se execută acțiune 2, ..., dacă are valoarea vn , se execută acțiune n ; altfel, se execută acțiune $n+1$. Fiecare valoare a selectorului corespunde unui **caz** tratat, căruia îi corespunde o acțiune care poate fi descrisă printr-un singur pas sau prin mai mulți pași.

Observați că acest tip de structură poate fi descris prin raționamentul **în cazul...** **caz1... caz2... altfel... sfârșit_în_cazul** (în **case... case1... case2... else... endcase**) pe care îl putem folosi în pseudocod pentru descrierea structurii.

în cazul că selector
 cazul v1: acțiune 1;
 cazul v2: acțiune 2;

 cazul vi: acțiune i;

 cazul vn: acțiune n;
 altfel acțiune n+1;
 sfârșit_în_cazul;

în case selector
 case v1: acțiune 1;
 case v2: acțiune 2;

 case vi: acțiune i;

 case vn: acțiune n;
 else acțiune n+1;
 endcase;

Cuvintele **în cazul că** (în **case**) marchează începutul structurii, iar cuvintele **sfârșit_în_cazul** (**endcase**) sfârșitul structurii. Cuvântul **cazul** (**case**) marchează începutul secvenței de pași care descriu acțiunea care se va executa pentru acel caz și cuvântul **altfel** (**else**) marchează începutul secvenței de pași care descriu acțiunea care se va executa dacă selectorul nu a avut valoarea nici unui caz.

Observație: Structura alternativă generalizată corespunde unor **structuri alternative simple imbricate**. O structură alternativă generalizată care are n cazuri este echivalentă cu n structuri alternative simple imbricate, în care condiția logică este dată de valoarea selectorului pentru acel caz.

```

întreg n,a,b,c;
real e;
început
  citește n,a,b,c;
  dacă n=1
    atunci e←(a+b)/c;
  altfel
    dacă n=2
      atunci e←(b+c)/a;
    altfel dacă n=3
      atunci e←(c+a)/b;
    sfârșit_dacă;
  sfârșit_dacă;
  scrie e;
sfârșit.
  
```

Exemplul a fost transpus în pseudocod.

Probleme care se pot rezolva cu ajutorul structurii alternative:

1. Se citesc patru numere întregi de la tastatură: **a**, **b**, **c** și **d**. Determinați care dintre produsele $a \times b$ și $c \times d$ este mai mare.
2. Se citesc trei numere **a**, **b** și **c**. Să se numere câte sunt pare.
3. Se citesc trei numere **a**, **b** și **c**. Să se verifice dacă ele pot fi termenii unei progresii aritmetice.
4. Se citesc trei numere **a**, **b** și **c**. Aflați dacă aceste numere pot reprezenta laturile unui triunghi. În caz afirmativ, calculați aria și afișați ce tip de triunghi este (oarecare, isoscel, echilateral, dreptunghic sau dreptunghic isoscel).
5. Se citește un număr întreg **n** care reprezintă un an calendaristic. Să se verifice dacă anul este bisect sau nu (condiția ca un an să fie bisect este ca, dacă anul nu este divizibil cu 100, să fie divizibil cu 4; altfel, să fie divizibil cu 400).
6. Se citesc două intervale de timp exprimate în ore, minute și secunde (**h1**, **m1** și **s1**, pentru primul interval, și **h2**, **m2** și **s2**, pentru al doilea interval). Să se calculeze suma celor două intervale de timp.
7. Se citesc trei numere reale. Să se calculeze minimul și maximul modulelor lor.
8. Se citesc două numere naturale **a** și **b**. Să se afișeze câte numere pare sunt în intervalul $[a, b]$.

```

întreg n,a,b,c;
real e;
început
  citește n,a,b,c;
  în cazul că n
    caz 1: e ← (a+b)/c;
    caz 2: e ← (b+c)/a;
    caz 3: e ← (c+a)/b;
  sfârșit în caz că;
  scrie e;
sfârșit.

```

3.2.3. Structura repetitivă

Prin această structură se execută repetat o acțiune sau o secvență de acțiuni, atât timp cât condiția precizată este adevărată:

- ✓ Cât timp mai sunt bilete, vindeți bilete; sau, vindeți bilete până le terminați.
- ✓ Cât timp semaforul este verde, mai trece o mașină.
- ✓ Cât timp mai aveți numere, le adunați la sumă.
- ✓ Cât timp mai aveți greșeli de corectat, corectați greșeli; sau, corectați greșeli până ați corectat și ultima greșeală.
- ✓ Cât timp mai aveți monede în buzunar, scoateți o monedă sau scoateți câte o monedă din buzunar până când nu mai aveți nici o monedă.
- ✓ Începând de la numărul 1, scrieți în ordine numerele până la 100.
- ✓ Începând de la numărul 2, scrieți în ordine numerele pare până la 50.

Executarea repetată a unei acțiuni sau a unei secvențe de acțiuni este un concept algoritmic foarte important. Metoda de implementare a unei astfel de repetiții este **structura repetitivă** sau **iterativă**, cunoscută sub numele de **ciclu** sau **bucă** (*loop*), în care un grup de acțiuni, numit **corpul ciclului** sau **iterație**, se execută repetat, sub un **proces de control** (testează condiție):


```

testează condiție
execută corpul ciclului
testează condiție
execută corpul ciclului
.....
testează condiție până când condiția nu mai este îndeplinită.

```

Exemplul 1

Se introduc de la tastatură mai multe numere, până când ultimul număr este 0, și trebuie să se calculeze suma numerelor. Se vor folosi două variabile de memorie: *s* (suma), care va avea inițial valoarea 0, și *a* (valoarea care se citește de la tastatură). Valoarea citită *a* se va aduna la sumă până când valoarea lui *a* va fi 0, adică putem spune **cât timp $a \neq 0$, adună-l pe *a* la *s***. Algoritmul va fi:

```

execută  $s \leftarrow 0$ 
citește valoarea lui a
testează  $a \neq 0$ 
execută  $s \leftarrow s + a$ ; citește a;
testează  $a \neq 0$ 
execută  $s \leftarrow s + a$ ; citește a;
.....
testează  $a \neq 0$  până când  $a = 0$  (condiția nu mai este îndeplinită)
scrie valoarea lui s

```

Exemplul 2

Să se calculeze suma a *n* numere introduse de la tastatură. Se vor folosi patru variabile de memorie: *n* (câte numere se citesc), *i* (un contor care numără câte numere s-au citit), care va avea inițial valoarea 1 (se citește primul număr), *s* (suma) care va avea inițial valoarea 0, și *a* (valoarea care se citește de la tastatură). Valoarea citită *a* se va aduna la sumă până când valoarea contorului *i* va fi mai mare decât *n*, adică putem spune **pentru $i \leq n$, adună-l pe *a* la *s***. Algoritmul va fi:

```

citește n; execută  $s \leftarrow 0$ ; execută  $i \leftarrow 1$ 
testează  $i \leq n$ 
citește a; execută  $s \leftarrow s + a$ ; execută  $i \leftarrow i + 1$ 
testează  $i \leq n$ 
citește a; execută  $s \leftarrow s + a$ ; execută  $i \leftarrow i + 1$ 
.....
testează  $i \leq n$  până când  $i > n$  (condiția nu mai este îndeplinită)
scrie valoarea lui s

```

Procesul de control cuprinde trei acțiuni:

Inițializarea. Stabilește starea inițială, starea dinainte de prima parcurgere a corpului ciclului. În primul exemplu, inițializarea cuprinde operația de atribuire $s \leftarrow 0$ și citirea primului număr (*citește a*). În al doilea exemplu, inițializarea cuprinde operațiile de atribuire $s \leftarrow 0$ și $i \leftarrow 1$.

Informatică

Testarea. Compară starea curentă cu starea care termină procesul de repetare și are rolul de a termina procesul de ciclare. Dacă cele două stări sunt egale, procesul de executare repetată a corpului ciclului se termină. În primul exemplu, se compară valoarea numărului citit de la tastatură, a , cu 0 ($a < 0$) și se continuă executarea repetată atât timp cât operatorul relațional furnizează valoarea „adevărat”. Procesul de executare repetată se termină atunci când valoarea lui a este 0. În al doilea exemplu, se compară valoarea contorului i cu numărul de executări repetate n ($i \leq n$) și se continuă ciclarea atât timp cât operatorul relațional furnizează valoarea „adevărat”. Procesul de executare repetată se termină atunci când valoarea lui i este mai mare ca n .

Modificarea. Schimbă starea curentă astfel încât să se avanseze către starea finală, care încheie procesul de repetare. Activitatea de modificare face parte din corpul ciclului. În primul exemplu, modificarea constă în citirea unei noi valori a lui a (citește a), care poate să fie 0, iar în exemplul al doilea modificarea constă în incrementarea cu 1 a contorului i ($i \leftarrow i+1$), prin care se evidențiază faptul că s-a mai citit un număr din cele n numere și că ne apropiem de citirea ultimului număr.

Inițializarea și modificarea sunt foarte importante, deoarece prin ele trebuie să se ajungă la condiția de terminare. De exemplu, dacă ciclarea se încheie atunci când o variabilă de memorie i are valoarea 20, și dacă inițializăm variabila de memorie i cu valoarea 0 și o modificăm prin incrementare cu 3, ea nu va avea niciodată valoarea 20, necesară terminării executării repetate a corpului ciclului: 0, 3, 6, 9, 12, 15, 18, 21, 24 etc. Un astfel de ciclu se numește **ciclu infinit** și presupune executarea unui număr infinit de pași, condiție care contrazice definiția unui algoritm.

Observați că cele două exemple diferă prin **informația pe care o avem referitor la numărul de executări repetate ale ciclului**. În primul exemplu, nu se cunoaște de câte ori se execută structura repetitivă (condiția de terminare este valoarea lui a , și nu se știe câte valori vor fi introduse pentru a până când a va avea valoarea 0). În al doilea exemplu, se cunoaște de câte ori se execută structura repetitivă (condiția de terminare este ca valoarea lui i , care pornește inițial de la valoarea 1, să ajungă printr-o incrementare cu 1 la valoarea n , deci să se execute de n ori corpul ciclului). În funcție de acest criteriu, există:

- structuri repetitive cu număr necunoscut de pași,
- structuri repetitive cu număr cunoscut de pași.

Structura repetitivă cu număr cunoscut de pași

În cazul **structurilor repetitive cu număr cunoscut de pași** sunt necesare două variabile de memorie: una numită **contor** (în exemplu, i), care se folosește pentru a număra de câte ori s-a executat repetat corpul ciclului, și una numită **număr de repetări** (în exemplu, n), care determină de câte ori trebuie să se execute repetat corpul ciclului. În acest caz, condiția de executare repetată a corpului ciclului este ca valoarea contorului să fie mai mică sau cel mult egală cu valoarea numărului de repetări. Inițializarea ciclului trebuie să conțină obligatoriu și pasul de inițializare a contorului, iar modificarea trebuie să conțină obligatoriu operația de incrementare a

contorului. Observați că acest tip de structură poate fi descris prin raționamentul **pentru... de la... până la... crescând cu... execută... sfârșit** (**for... to... by... do... endfor**) pe care îl putem folosi în pseudocod pentru descrierea structurii.

```

pentru contor ← vi, vf [pas v] execută
    acțiune;
sfârșit_pentru;
  
```

```

for contor ← vi, vf [by v] do
    acțiune;
endfor;
  
```

Cuvântul **pentru** (**for**) marchează începutul structurii iar cuvântul **sfârșit_pentru** (**endfor**) sfârșitul structurii. Virgula dintre *contor* = *vi* și *vf* delimitează operația de atribuire prin care se face inițializarea contorului (*vi*), de valoarea finală la care trebuie să ajungă contorul (*vf*), cuvântul **pas** (**by**) precede valoarea cu care se incrementează contorul (*v*), iar cuvântul **execută** (**do**) marchează începutul secvenței de pași care se va executa repetat. Dacă incrementarea contorului se face cu valoarea 1, expresia **pas v** (**by v**) este implicită și nu mai trebuie menționată.

Exemplul 2 a fost transpus în pseudocod.

```

întreg i, n;
real s, a;
început
    citește n;
    s ← 0;
    pentru i ← 1, n execută
        citește a;
        s ← s + a;
    sfârșit_pentru;
    scrie s;
sfârșit.
  
```

Probleme care se pot rezolva cu ajutorul structurii repetitive cu număr cunoscut de pași:

1. Se citesc două numere naturale *m* și *n*. Calculați n^m .
2. Se citesc de la tastatură *n* numere întregi. Să se afle câte numere sunt negative.
3. Se citesc de la tastatură *n* numere întregi. Să se calculeze media aritmetică a numerelor pare.
4. Să se afișeze primele *n* numere naturale divizibile cu 5.
5. S-a depus într-un cont la o bancă o sumă de *s* unități monetare. Banca practică următoarele dobânzi: **d1** – lunar, **d2** – trimestrial, **d3** – semestrial și **d4** – anual. Calculați ce sumă va fi în cont după *n* ani, dacă se capitalizează dobânda în toate cele patru cazuri. Ce dobândă este mai avantajoasă?
6. Se citesc două numere întregi, *a* și *b*. Să se calculeze produsul *a* × *b* fără a folosi operatorul pentru înmulțire (**Indicație**. Rezultatul se va obține prin adunarea repetată a lui *a* de *|b|* ori și se va face discuție după semnul operanzilor).
7. Se citesc două numere întregi, *a* și *b*. Să se calculeze câtul și restul împărțirii lui *a* la *b*, fără a folosi operatorii **mod** și **div** (**Indicație**. Rezultatul se va obține prin scăderea repetată a lui *|b|* din *|a|* și se va face discuție după semnul operanzilor).
8. Se citesc de la tastatură *n* numere întregi nenule. Să se calculeze suma celor de rang par și produsul celor de rang impar.
9. Se citește un număr natural *n*. Să se determine toate perechile de numere naturale *a* și *b* care verifică relația $a^2 + b^2 = n$.
10. Calculați suma:

$$S = 1 \times 3 + 2 \times 5 + 3 \times 7 + \dots + n \times (2n+1)$$

unde $n \in \mathbb{N}$, valoarea lui introducându-se de la tastatură.

11. Calculați suma:

$$S = 1 + 1 \times 2 + 1 \times 2 \times 3 + \dots + 1 \times 2 \times 3 \times \dots \times n$$

unde $n \in \mathbb{N}$, valoarea lui introducându-se de la tastatură.

12. Calculați suma:

$$S = 1^2 - 2^2 + 3^2 - 4^2 + \dots + (-1)^{n+1} \times n^2$$

unde $n \in \mathbb{N}$, valoarea lui introducându-se de la tastatură.

13. Calculați suma:

$$S = 1 - a + a^2 - a^3 + a^4 - \dots + (-1)^n \times a^n$$

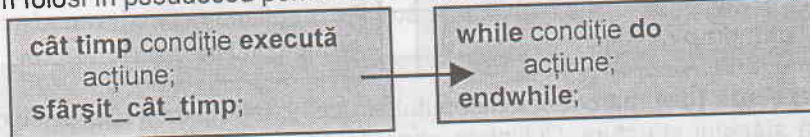
unde $a \in \mathbb{R}$ și $n \in \mathbb{N}$, valorile lor introducându-se de la tastatură.

Structura repetitivă cu număr necunoscut de pași

În funcție de momentul în care se face testarea, există două tipuri de structuri repetitive cu număr necunoscut de pași:

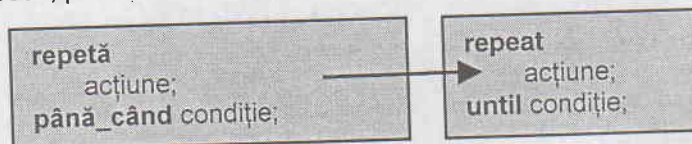
- structuri repetitive condiționate anterior,
- structuri repetitive condiționate posterior.

Structura repetitivă condiționată anterior testează condiția de terminare a ciclului înainte de executarea corpului ciclului. Acest tip de structură poate fi descris prin raționamentul **cât timp... execută... sfârșit (while... do.. endwhile)** pe care îl putem folosi în pseudocod pentru descrierea structurii.



Cuvântul **cât timp (while)** marchează începutul structurii, iar cuvântul **sfârșit_cât_timp (endwhile)** – sfârșitul structurii. Cuvintele **cât timp (while)** și **execută (do)** delimitează expresia logică ce se evaluează pentru a testa condiția de terminare a ciclului (corpul ciclului se execută dacă expresia logică are valoarea „adevărat”), iar cuvântul **execută (do)** marchează începutul secvenței de pași care se vor executa repetat.

Structura repetitivă condiționată posterior testează condiția de terminare a ciclului după executarea corpului ciclului. Acest tip de structură poate fi descris prin raționamentul **repetă... până când... (repeat... until...)** pe care îl putem folosi în pseudocod, pentru descrierea structurii.



Cuvântul **repetă (repeat)** marchează începutul structurii, iar sfârșitul expresiei logice marchează sfârșitul structurii. Cuvintele **repetă (repeat)** și **până_când (until)** delimitează secvența de pași care se va executa repetat, iar cuvântul **până_când**

(until) precede expresia logică ce se evaluează pentru a testa condiția de terminare a ciclului (ciclul se termină când condiția logică are valoarea *adevărat*, altfel spus, corpul ciclului se execută cât timp expresia logică are valoarea *fals*).

Același algoritm iterativ poate fi descris prin ambele tipuri de structuri repetitive. Exemplul 1 a fost transpus în pseudocod și schemă logică prin cele două tipuri de structuri.

Structura repetitivă condiționată anterior	Structura repetitivă condiționată posterior
<pre> real s,a; început s ← 0; citește a; cât timp a<>0 execută s ← s+a; citește a; sfârșit_cât_timp; scrie s; sfârșit. </pre>	<pre> real s,a; început s ← 0; citește a; repetă s ← s+a; citește a; până_când a=0; scrie s; sfârșit. </pre>

Structura repetitivă condiționată posterior mai poate fi descrisă și prin raționamentul **execută... cât timp ... (do... while...)** pe care îl putem folosi în pseudocod, pentru descrierea structurii.



Cuvântul **execută (do)** marchează începutul structurii, iar sfârșitul expresiei logice marchează sfârșitul structurii. Cuvintele **execută (do)** și **cât timp (while)** delimitează secvența de pași care se va executa repetat, iar cuvântul **cât timp (while)** precede expresia logică ce se evaluează pentru a testa condiția de terminare a ciclului (ciclul se execută cât timp expresia logică are valoarea *adevărat*, altfel spus execuția sa se termină când expresia logică are valoarea *fals*).

Structura repetitivă cât timp ... sfârșit_cât_timp	Structura repetitivă execută... cât timp ...
<pre> real s,a; început s ← 0; citește a; cât timp a<>0 execută s ← s+a; citește a; sfârșit_cât_timp; scrie s; sfârșit. </pre>	<pre> real s,a; început s ← 0; citește a; execută s ← s+a; citește a; cât timp a<>0; scrie s; sfârșit. </pre>

Observații:

1. Pentru același algoritm, cele două condiții de testare a terminării executării repetate folosite de cele două tipuri de structuri repetitive **cât timp** și **repetă** sunt complementare. În exemplu, $\text{not}(a < 0) = (a = 0)$. Dacă notăm cu $c1$ condiția structurii **cât timp** și cu $c2$ condiția structurii **repetă**, atunci:

$$\text{not}(c1) = c2$$

2. Pentru același algoritm, cele două condiții de testare a terminării executării repetate folosite de structurile repetitive **cât timp** și **execută** sunt aceleași. În exemplu $(a < 0) = (a < 0)$. Dacă notăm cu $c1$ condiția structurii **cât timp** și cu $c2$ condiția structurii **execută**, atunci:

$$c1 = c2$$

3. În cazul structurii condiționate posterior este obligatoriu să se execute acțiunea cel puțin o dată, spre deosebire de structura repetitivă condiționată anterior, unde este posibil să nu se execute niciodată (în exemplu, cazul în care prima valoare citită pentru a este 0).

Concluzii:

Orice structură repetitivă condiționată anterior poate fi transformată într-o structură repetitivă condiționată posterior. Reciproca este și ea adevărată.

Dacă vom considera că în corpul ciclului există două tipuri de acțiuni: una de modificare, care schimbă starea curentă astfel încât să se avanseze către starea finală (*acțiune_modificare*), și una prin care se efectuează prelucrările obișnuite (*acțiune_prelucrare*), atunci următoarele două structuri sunt echivalente:

Structura repetitivă condiționată anterior	Structura repetitivă condiționată posterior
<pre> acțiune_inițializare; cât timp condiție execută acțiune_prelucrare; acțiune_modificare; sfârșit_cât_timp;</pre>	<pre> acțiune_inițializare; repetă acțiune_prelucrare; acțiune_modificare; până_când not condiție;</pre>

Orice structură repetitivă cu număr cunoscut de pași poate fi transformată într-o structură repetitivă cu număr necunoscut de pași. Reciproca nu este adevărată.

Structura repetitivă cu număr cunoscut de pași	Structura repetitivă cu număr necunoscut de pași condiționată posterior
<pre> pentru contor ← vi, vf execută acțiune_prelucrare; sfârșit_pentru;</pre>	<pre> contor ← vi; cât timp contor <= vf acțiune_prelucrare; contor ← contor + 1; sfârșit_cât_timp;</pre>

Probleme care se pot rezolva cu ajutorul structurii repetitive cu număr necunoscut de pași:

1. Se citesc mai multe numere până când ultimul număr citit este zero. Să se afle câte numere sunt pozitive și câte numere sunt negative.
2. Se citesc mai multe numere întregi până când ultimul număr citit este zero. Să se calculeze media aritmetică a numerelor impare.
3. Se citește un număr natural n . Să se afișeze toate numerele naturale mai mici decât n care sunt divizibile cu 3.
4. Se citesc mai multe numere întregi până când ultimul număr citit este zero. Să se calculeze suma celor de rang par și produsul celor de rang impar.

3.3. Algoritmi elementari

Ați văzut că cea mai importantă etapă în rezolvarea unei probleme cu ajutorul calculatorului este cea de stabilire a algoritmului, deci de găsim a metodei de rezolvare a problemei a cărei soluție urmează să fie calculată. Pentru a vă ușura munca de descoperire a algoritmului, puteți să vă folosiți de **algoritmi elementari**. Acești algoritmi oferă metode de rezolvare pentru probleme clasice:

- ✓ algoritmi pentru interschimbarea valorilor a două numere;
- ✓ algoritmi pentru determinarea valorii minime (maxime);
- ✓ algoritmi pentru prelucrarea cifrelor unui număr;
- ✓ algoritmi pentru determinarea c.m.m.d.c. dintre două numere;
- ✓ algoritmi pentru testarea numerelor prime;
- ✓ algoritmi pentru prelucrarea divizorilor unui număr;
- ✓ algoritmi pentru conversii între sisteme de numerație;
- ✓ algoritmi pentru generarea șirurilor recurente.

Cu ajutorul acestor algoritmi puteți prelucra un număr sau o mulțime de valori numerice pentru a obține informații.

3.3.1. Algoritmi pentru interschimbare

Interschimbarea valorilor a două variabile de memorie x și y nu se poate face prin simpla atribuire a noii valori, deoarece secvența de atribuiri $x \leftarrow y$; și $y \leftarrow x$; duce la pierderea valorii lui x , iar secvența de atribuiri $y \leftarrow x$; și $x \leftarrow y$; duce la pierderea valorii lui y . Pentru a realiza interschimbarea, puteți folosi una dintre următoarele variante de algoritmi:

Varianta 1. Interschimbarea valorilor a două variabile (a și b) prin folosirea unei **variabile intermediare** (x). Variabila intermediară se folosește pentru salvarea valorii care se distruge prin prima operație de atribuire. Pașii algoritmului sunt:

- Pasul 1.** Se salvează valoarea primei variabile (a) în variabila x , prin $x \leftarrow a$.
- Pasul 2.** Se atribuie primei variabile, a cărei valoare a fost salvată (a), valoarea celei de a doua variabile (b), prin $a \leftarrow b$.

Pasul 3. Se atribuie celei de a doua variabile (b) valoarea primei variabile care a fost salvată (x), prin $b \leftarrow x$.

Varianta 2. Interschimbarea valorilor a două variabile (a și b) fără folosirea unei variabile intermediare. Se folosesc identitățile matematice $a = (a-b)+b$ și $b = ((a-b)+b)-(a-b)$. Pentru interschimbarea valorilor se folosește valoarea $a-b$, care va fi atribuită inițial variabilei a .

Varianta 1

```
real a,b,x;
inceput
citește a,b;
x ← a;
a ← b;
b ← x;
scrie a,b;
sfârșit.
```

Varianta 2

```
real a,b;
inceput
citește a,b;
a ← a-b;
b ← a+b;
a ← b-a;
scrie a,b;
sfârșit.
```

Studiu de caz

Scop: exemplificarea modului în care poate fi folosit algoritmul de interschimbare în rezolvarea unei probleme.

Enunțul problemei: Se citește un număr natural format din 3 cifre. Să se afișeze numărul minim care se poate forma din cifrele sale. De exemplu, dacă numărul este 312, numărul minim care se poate forma cu cifrele sale este 123.

Algoritmul constă în executarea următorilor pași:

Pasul 1. Se extrag cifrele numărului.

Pasul 2. Se ordonează crescător cifrele numărului.

Pasul 3. Se formează un nou număr din cifrele ordonate crescător.

Pentru ordonarea crescătoare a cifrelor, se va rafina **Pasul 2**. **Procesul de rafinare** constă în descompunerea unei probleme în subprobleme.

Considerând cifrele numărului: a (cifra sutelor), b (cifra zecilor) și c (cifra unităților), pentru a le ordona le vom compara două câte două și, dacă nu respectă relația de ordine precizată, le interschimbăm:

Pasul 2.1. Se compară a cu b . Dacă a este mai mare decât b , se interschimbă a cu b .

Pasul 2.2. Se compară b cu c . Dacă b este mai mare decât c , se interschimbă b cu c .

Pasul 2.3. Se compară a cu b . Dacă a este mai mare decât b , se interschimbă a cu b .

Se vor folosi următoarele variabile de memorie:

- **Data de intrare** n pentru număr.
- **Date intermediare** a , b și c , pentru cifrele numărului, și x pentru operația de interschimbare.
- **Data de ieșire** n pentru noul număr (putem folosi aceeași variabilă de memorie, deoarece nu mai avem nevoie de vechea valoare a numărului).

Algoritmul este:

```

întreg n, a, b, c, x;
inceput
  citește n;
  a ← n div 100; b ← (n div 10) mod 10; c ← n mod 10;
  dacă a > b
    atunci x ← a; a ← b; b ← x;
  sfârșit dacă
  dacă b > c
    atunci x ← b; b ← c; c ← x;
  sfârșit dacă
  dacă a > b
    atunci x ← a; a ← b; b ← x;
  sfârșit dacă
  n ← a*100+b*10+c;
  scrie n;
sfârșit.

```

Pentru testarea algoritmului folosiți următoarea mulțime de valori pentru data de intrare n: {123, 132, 213, 231, 312, 321}. Testați algoritmul.

Refaceți algoritmul folosind metoda de interschimbare care nu utilizează o variabilă intermediară. Testați algoritmul.

Probleme care se pot rezolva cu ajutorul algoritmului de interschimbare:

1. Se citesc trei numere întregi a, b și c. Dacă numerele sunt diferite de zero, să se afișeze media lor geometrică; altfel, să se afișeze numerele în ordine descrescătoare.
2. Se citește un număr natural format din 3 cifre. Să se afișeze numărul maxim care se poate forma din cifrele sale.
3. Se citește un număr natural format din 4 cifre. Să se afișeze numărul maxim și numărul minim care se pot forma din cifrele sale.

3.3.2. Algoritm pentru determinarea maximului (minimului)

Algoritmul determină valoarea maximă (minimă) dintr-un șir de numere introduse de la tastatură. Algoritmul constă în atribuirea valorii primului element maximului (minimului) și compararea acestei valori cu elementele din șir. Pașii care se execută sunt:

- Pasul 1.** Se citește primul număr a.
- Pasul 2.** Se atribuie maximului valoarea lui a, prin $\max \leftarrow a$.
- Pasul 3.** Se citește următorul număr a.
- Pasul 4.** Dacă $a > \max$, atunci se atribuie maximului valoarea lui a, prin $\max \leftarrow a$.
- Pasul 5.** Dacă mai sunt date de citit, se revine la **Pasul 3**.

Se vor folosi următoarele variabile de memorie:

- ✓ **Data de intrare** a pentru citirea unui număr.
- ✓ **Data de ieșire** max pentru valoarea maximă.

Varianta 1. Se introduce un șir de n numere de la tastatură. Să se afișeze maximul dintre aceste numere.

Varianta 2. Se introduce un șir de numere de la tastatură până la întâlnirea valorii 0. Să se afișeze maximul dintre aceste numere.

Varianta 1

```

intreg a,max,n,i;
inceput
  citește n,a;
  max ← a;
  pentru i ← 2,n execută
    citește a;
    dacă a>max
      atunci max ← a;
    sfârșit_dacă;
  sfârșit_pentru;
  scrie max;
sfârșit.

```

Varianta 2

```

intreg a,max;
inceput
  citește a;
  max ← a;
  cât timp a<>0 execută
    dacă a>max
      atunci max ← a;
    sfârșit_dacă;
    citește a;
  sfârșit_cât timp;
  scrie max;
sfârșit.

```

Studiu de caz

Scop: exemplificarea modului în care poate fi folosit algoritmul pentru determinarea valorii maxime (minime).

Enunțul problemei: Se introduc de la tastatură n numere. Să se afișeze valoarea maximă și de câte ori apare în șir.

Pentru numărarea aparițiilor maximului se folosește variabila de memorie k .

Pasul 1. Se citește primul număr a .

Pasul 2. Se atribuie maximului valoarea lui a , prin $\text{max} \leftarrow a$, și se inițializează contorul k , prin $k \leftarrow 1$.

Pasul 3. Se citește următorul număr a .

Pasul 4. Dacă $a = \text{max}$, atunci se incrementează contorul k , prin $k \leftarrow k + 1$.

Pasul 5. Dacă $a > \text{max}$, atunci se atribuie maximului valoarea lui a , prin $\text{max} \leftarrow a$, și se inițializează contorul k , prin $k \leftarrow 1$.

Pasul 6. Dacă mai sunt date de citit, se revine la **Pasul 3**.

```

intreg a,max,n,i;
inceput
  citește n,a;
  max ← a; k ← 1;
  pentru i ← 2,n execută
    citește a;
    dacă a=max
      atunci k ← k+1;
    altfel dacă a>max
      atunci max ← a; k ← 1;
    sfârșit_dacă;
  sfârșit_dacă;

```



```
sfârșit pentru;
scrie max, k;
sfârșit.
```



Probleme care se pot rezolva cu ajutorul algoritmilor de determinare a minimului (maximului)

1. Se introduc de la tastatură n numere. Să se afișeze valoarea minimă și valoarea maximă.
2. Se introduce un șir de numere de la tastatură, până la întâlnirea valorii 0. Să se afișeze maximul și minimul dintre aceste numere.
3. La un concurs, comisia de notare este formată din n membri. Să se scrie algoritmul de calcul al mediei, știind că nota cea mai mică și nota cea mai mare nu sunt luate în considerare la calcularea mediei.
4. Se introduce un șir de numere de la tastatură, până la întâlnirea valorii 0. Să se afișeze valoarea maximă și de câte ori apare în șir.
5. Se introduce un șir de numere întregi de la tastatură, până la întâlnirea valorii 0. Să se afișeze:
 - a) maximul dintre numerele negative;
 - b) minimul dintre numerele negative;
 - c) maximul dintre numerele pozitive;
 - d) minimul dintre numerele pozitive.
6. Se introduce un șir de n numere întregi de la tastatură. Să se afișeze:
 - a) maximul dintre numerele negative;
 - b) minimul dintre numerele negative;
 - c) maximul dintre numerele pozitive;
 - d) minimul dintre numerele pozitive.
7. Se introduc de la tastatură n numere întregi. Să se afișeze primele două valori maxime (sau, variantă, minime) și de câte ori apar în șirul de numere.

3.3.3. Algoritmi pentru prelucrarea cifrelor unui număr

Pentru prelucrarea cifrelor unui număr puteți folosi următorii algoritmi;

- ✓ algoritmul pentru extragerea cifrelor unui număr;
- ✓ algoritmul pentru compunerea unui număr din cifrele sale;
- ✓ algoritmul pentru determinarea inversului unui număr (inversarea cifrelor unui număr).

Algoritmul pentru extragerea cifrelor unui număr

Algoritmul determină cifrele unui număr n , prin extragerea pe rând a fiecărei cifre c (începând cu cifra unităților), cu operația $n \bmod 10$, și eliminarea din număr a cifrei extrase, cu operația $n \div 10$. Aceste operații se execută cât timp mai există cifre de extras din n ($n \neq 0$). Pașii care se execută sunt:

Pasul 1. Se extrage cifra cea mai nesemnificativă, cu operația $c \leftarrow n \bmod 10$.

Pasul 2. Se afișează (se prelucrează) cifra.



Pasul 3. Se elimină din număr cifra extrasă, cu operația $n \leftarrow n \text{ div } 10$.

Pasul 4. Dacă $n \neq 0$, atunci se revine la **Pasul 1**.

Se vor folosi următoarele variabile de memorie:

- ✓ **Data de intrare** n pentru numărul care se citește.
- ✓ **Data de ieșire** c pentru cifra numărului.

```

intreg n, c;
inceput
  citește n;
  cât timp  $n \neq 0$  execută
     $c \leftarrow n \bmod 10$ ;
    scrie c;
     $n \leftarrow n \text{ div } 10$ ;
  sfârșit_cât_timp;
sfârșit.

```

Studiu de caz

Scop: exemplificarea modului în care poate fi folosit algoritmul pentru extragerea cifrelor unui număr.

Enunțul problemei 1: Se citește un număr natural n . Să se afișeze suma și produsul cifrelor sale.

Se vor folosi următoarele variabile de memorie:

- ✓ **Data de intrare:** n , pentru numărul care se citește.
- ✓ **Date de ieșire:** s , pentru suma cifrelor, și p , pentru produsul cifrelor numărului.

Enunțul problemei 2: Se introduce de la tastatură un șir de numere naturale, până la citirea valorii 0. Să se afișeze toate perechile de numere introduse consecutiv care au proprietatea că al doilea număr este egal cu suma cifrelor primului număr.

Se vor folosi următoarele variabile de memorie;

- ✓ **Date de intrare:** a și b , pentru perechile de numere citite consecutiv.
- ✓ **Date intermediare:** s pentru calcularea sumei cifrelor și x pentru salvarea valorii lui a .
- ✓ **Date de ieșire:** perechile de numere a și b care îndeplinesc condiția.

Algoritmii sunt:

Problema 1

```

intreg n, s, p;
inceput
  citește n;
   $s \leftarrow 0$ ;  $p \leftarrow 1$ ;
  cât timp  $n \neq 0$  execută
     $s \leftarrow s + n \bmod 10$ ;
     $p \leftarrow p * (n \bmod 10)$ ;
     $n \leftarrow n \text{ div } 10$ ;
  sfârșit_cât_timp;
  scrie s, p;
sfârșit.

```

Problema 2

```

intreg a, b, s, x;
inceput
  citește a, b;
  cât timp  $b \neq 0$  execută
     $s \leftarrow 0$ ;  $x \leftarrow a$ ;
    cât timp  $x \neq 0$  execută
       $s \leftarrow s + x \bmod 10$ ;
       $x \leftarrow x \text{ div } 10$ ;
    sfârșit_cât_timp;
    dacă  $s = b$ 
      atunci scrie a, b;

```



```
sfârșit dacă;
a ← b;
citește b;
sfârșit_cât_timp;
sfârșit.
```

Alegeți, pentru testarea algoritmilor, câte o mulțime completă de seturi de date de intrare. Testați algoritmii.



Algoritmul pentru compunerea unui număr din cifrele sale

Citirea cifrelor numărului se face începând cu cifra cea mai semnificativă. Algoritmul folosește reprezentarea numărului în baza 10:

$$a_n a_{n-1} \dots a_1 a_0 = a_n \times 10^n + a_{n-1} \times 10^{n-1} + \dots + a_1 \times 10^1 + a_0 \times 10^0$$

Varianta 1. Se introduc pe rând de la tastatură cele n cifre ale unui număr, începând cu cifra cea mai semnificativă. Să se afișeze numărul nr obținut din aceste cifre. Pașii algoritmului sunt:

Pasul 1. Se citește numărul de cifre n .

Pasul 2. Se calculează p ca fiind 10^{n-1} astfel:

Pasul 2.1. Se inițializează p , cu valoarea 1, prin operația $p \leftarrow 1$.

Pasul 2.2. Se inițializează contorul i , cu care se numără puterea lui 10, cu valoarea 1, prin operația $i \leftarrow 1$.

Pasul 2.3. Se înmulțește p cu valoarea 10, prin operația $p \leftarrow p \times 10$.

Pasul 2.4. Se incrementează contorul i cu 1, prin operația $i \leftarrow i + 1$.

Pasul 2.5. Dacă $i \leq n-1$, se revine la pasul 2.3.

Pasul 3. Se inițializează numărul nr cu valoarea 0, prin operația $nr \leftarrow 0$.

Pasul 4. Se inițializează contorul i , cu care se numără cifrele citite, cu 1, prin $i \leftarrow 1$.

Pasul 5. Se citește cifra c .

Pasul 6. Se adună la numărul nr cifra c înmulțită cu puterea lui 10 corespunzătoare poziției ei în număr, prin operația $nr \leftarrow nr + c \times p$.

Pasul 7. Se decrementează puterea lui 10, prin operația $p \leftarrow p \div 10$.

Pasul 8. Se incrementează contorul i cu 1, prin operația de atribuire $i \leftarrow i + 1$.

Pasul 9. Dacă $i \leq n$, atunci se revine la **Pasul 5**.

Pasul 10. Se afișează numărul nr .

Varianta 1. Se introduc pe rând, de la tastatură, mai multe numere care reprezintă cifrele unui număr, până când se introduce un număr care nu poate fi cifră. Să se afișeze numărul nr obținut din aceste cifre. Pentru rezolvarea problemei, algoritmul folosește următorul mod de grupare a termenilor reprezentării numărului în baza 10:

$$a_n a_{n-1} \dots a_1 a_0 = (\dots((a_n \times 10 + a_{n-1}) \times 10 + a_{n-2}) \times 10 + \dots + a_1) \times 10 + a_0$$

Pașii care se execută sunt:

Pasul 1. Se inițializează numărul nr cu valoarea 0, prin operația $nr \leftarrow 0$.

Pasul 2. Se citește cifra c .

Pasul 3. Se adună la numărul nr înmulțit cu 10 cifra citită, prin $nr \leftarrow nr \times 10 + c$.

Pasul 4. Se citește cifra c .

Pasul 5. Dacă c este o cifră ($c \geq 0$ and $c \leq 9$), atunci se revine la **Pasul 3**.

Pasul 6. Se afișează numărul *nr*.

Varianta 1

```

intreg n,p,nr,i;
inceput
  citește n;
  p ← 1;
  pentru i ← 1,n-1 execută
    p ← p*10;
  sfârșit_pentru;
  nr ← 0;
  pentru i ← 1,n execută
    citește c;
    nr ← nr + c*p;
    p ← p div 10;
  sfârșit_pentru;
  scrie nr;
sfârșit.

```

Varianta 2

```

intreg c,nr;
inceput
  nr ← 0;
  citește c;
  cât timp c>=0 and c<=9 execută
    nr ← nr*10 + c;
    citește c;
  sfârșit_cât_timp;
  scrie nr;
sfârșit.

```

Studiu de caz

Scop: exemplificarea modului în care poate fi folosit algoritmul pentru compunerea unui număr din cifrele sale.

Enunțul problemei: Se citesc mai multe numere, care reprezintă cifrele unui număr binar, până când numărul citit nu mai este cifră binară. Să se afișeze numărul binar. De exemplu, dacă se introduc numerele 1, 0, 1, 1, 5, numărul binar afișat va fi 1011.

Pentru a scrie pe ecran numărul obținut, se va folosi scrierea lui ca un număr în baza 10 (cu cifrele corespunzătoare reprezentării în baza 2).

```

intreg c,nr;
inceput
  nr ← 0;
  citește c;
  cât timp c=0 or c=1 execută
    nr ← nr*10 + c;
    citește c;
  sfârșit_cât_timp;
  scrie nr;
sfârșit.

```

Algoritmul pentru determinarea inversului unui număr

Algoritmul determină *inv*, inversul numărului *n*, prin extragerea pe rând a fiecărei cifre începând cu cifra unităților) din numărul *n* și compunerea unui nou număr cu aceste cifre. De exemplu, dacă numărul este 123, inversul va fi 321. Pașii algoritmului sunt:

Pasul 1. Se citește numărul *n*.

Pasul 2. Se inițializează numărul invers *inv* cu valoarea 0, prin operația $inv \leftarrow 0$.

- Pasul 3.** Se extrage cifra cea mai nesemnificativă din numărul n și se adună cifra la numărul inv înmulțit cu 10, prin operația $inv \leftarrow inv * 10 + n \bmod 10$.
- Pasul 4.** Se elimină din numărul n cifra extrasă, cu operația $n \leftarrow n \div 10$.
- Pasul 5.** Dacă $n < 0$, atunci se revine la **Pasul 3**.

```

intreg n, inv;
inceput
citește n;
inv ← 0;
cât timp n <> 0 execută
    inv ← inv*10 + n mod 10;
    n ← n div 10;
sfârșit cât timp;
scrie inv;
sfârșit.

```

Studiu de caz

Scop: exemplificarea modului în care poate fi folosit algoritmul pentru determinarea inversului unui număr.

Enunțul problemei: Se citește un număr natural n . Să se verifice dacă este **palindrom** (un număr este palindrom dacă, citit de la stânga la dreapta și de la dreapta la stânga, are aceeași valoare; de exemplu, 12321).

Deoarece, prin calcularea inversului numărului n , valoarea acestuia se pierde prin eliminarea cifrelor, iar după calcularea inversului mai avem nevoie de valoarea lui n pentru a o compara cu valoarea inversului, se va folosi o variabilă de memorie nr pentru a salva valoarea lui n .

```

intreg n, nr, inv;
inceput
citește n;
nr ← n; inv ← 0;
cât timp n <> 0 execută
    inv ← inv*10 + n mod 10; n ← n div 10;
sfârșit cât timp;
dacă nr=inv
    atunci scrie "Numarul este palindrom";
    altfel scrie "Numarul nu este palindrom";
sfârșit dacă;
sfârșit.

```



Probleme care se pot rezolva cu ajutorul algoritmilor pentru prelucrarea cifrelor unui număr:

1. Se citește un număr natural n . Să se afișeze suma și produsul cifrelor pare (sau impare).
2. Se citește un număr natural n . Să se afișeze suma și produsul cifrelor din pozițiile pare (sau, variantă, impare). Numărarea pozițiilor se face începând cu cifra cea mai semnificativă.

Informatică

3. Se introduc de la tastatură n numere. Să se afișeze cea mai mare cifră a fiecărui număr.
4. Să se afișeze toate numerele naturale care au proprietatea că sunt egale cu pătratul sumei cifrelor lor (Indicație. Se demonstrează matematic că un astfel de număr nu poate avea decât maxim 4 cifre. Exemplu: $81 \Rightarrow 8+1=9$; $81=9 \times 9$).
5. Să se afișeze toate numerele naturale mai mici decât n care au proprietatea că pătratul și cubul fiecăruia au cel puțin o cifră comună.
6. Să se afișeze toate numerele naturale mai mici decât n care au proprietatea că pătratul și cubul fiecăruia au cel puțin o cifră comună; pentru fiecare număr găsit să se afișeze câte cifre sunt comune și care sunt acelea.
7. Se citește n numere naturale. Să se afișeze, pentru fiecare număr din șir, numărul obținut prin eliminarea tuturor cifrelor 0.
8. Să se afișeze toate numerele care sunt palindrom și care aparțin intervalului $[a, b]$. Valorile pentru a și b se citește de la tastatură.
9. Se citește un șir de n numere naturale. Să se afișeze cele care sunt palindroame.
10. Să se afișeze toate numerele din intervalul $[a, b]$ care au suma cifrelor un număr par. Valorile pentru a și b se citește de la tastatură.
11. Se citește un număr natural. Să se afișeze inversul sumei cifrelor sale.
12. Se citește de la tastatură un număr $k \neq 0$ și un șir de numere întregi, până la întâlnirea numărului 0. Să se afișeze câte numere din șir au suma cifrelor k .
13. Să se găsească toate numerele de două cifre care au proprietatea că inversul pătratului fiecăruia este egal cu pătratul inversului.
14. Se introduce de la tastatură un șir de n numere naturale. Să se afișeze câtul și restul împărțirii dintre suma numerelor și suma cifrelor sumei numerelor.
15. Se introduce de la tastatură un șir de numere naturale, până la citirea numărului 0. Să se afișeze toate tripletele de numere introduse consecutiv care au proprietatea că al doilea și al treilea număr sunt egale cu câtul, respectiv cu restul dintre împărțirea primului număr la suma cifrelor sale.
16. Se citește un număr natural n . Să se afișeze toate numerele mai mici decât n care sunt egale cu suma pătratelor cifrelor lor.
17. Se citește de la tastatură un număr $k \in [0, 9]$ și un șir de numere naturale, până la citirea numărului 0. Să se afișeze toate perechile de numere introduse consecutiv care au proprietatea că au același număr de apariții ale cifrei k în pătratul lor.
18. Se introduce de la tastatură un șir de numere naturale până la citirea numărului 0. Să se afișeze toate perechile de numere introduse consecutiv care au proprietatea că suma cifrelor primului număr este pară, iar suma cifrelor celui de al doilea număr este impară.
19. Se citește un număr natural n . Să se afișeze numărul obținut prin eliminarea cifrei din mijloc, dacă n are un număr impar de cifre, respectiv a celor două cifre din mijloc, dacă n are un număr par de cifre.
20. Să se calculeze suma tuturor numerelor formate din cifre impare distincte. Indicație: cel mai mic număr este 1, iar cel mai mare 97531.



3.3.4. Algoritmi pentru calcularea c.m.m.d.c.

Pentru calcularea c.m.m.d.c dintre două numere naturale nenule, se folosesc următorii algoritmi:

Varianta 1. Folosește **algoritmul lui Euclid**, care **atribuie** lui **b** **restul** împărțirii lui **a** la **b**, iar lui **a** vechea valoare a lui **b**. Rezolvarea problemei se bazează pe condiția $b \neq 0$. Pașii algoritmului sunt:

Pasul 1. Se împarte **a** la **b** și se obține restul **r** ($r \leftarrow a \bmod b$).

Pasul 2. Se execută operațiile de atribuire $a \leftarrow b$; $b \leftarrow r$.

Pasul 3. Dacă $b <> 0$, atunci se revine la **Pasul 1**; altfel, $\text{cmmdc} \leftarrow a$.

De exemplu, dacă $a=18$ și $b=12$, calculul se desfășoară astfel:

1. Se calculează restul: $r = 18 \bmod 12 = 6$.
2. Se fac atribuirile: $a=12$, $b=6$.
3. $b \neq 0$ ($6 \neq 0$) $\Rightarrow$ Se calculează restul: $r = 12 \bmod 6 = 0$.
4. Se fac atribuirile: $a=6$, $b=0$.
5. $b=0$ ($0=0$) $\Rightarrow \text{cmmdc} = a = 6$.

Verificați algoritmul și pentru $a=12$ și $b=18$.

Varianta 2. Folosește **algoritmul de scădere repetată** a valorii celei mai mici din valoarea cea mai mare. Rezolvarea problemei se bazează pe condiția $a \neq b$. Pașii care se execută sunt:

Pasul 1. Se scade, din numărul mai mare, celălalt număr: dacă $a > b$, se execută operația $a \leftarrow a - b$; altfel, se execută operația $b \leftarrow b - a$.

Pasul 2. Dacă $a <> b$, atunci se revine la **Pasul 1**, altfel $\text{cmmdc} \leftarrow a$.

De exemplu, dacă $a=18$ și $b=12$, calculul se desfășoară astfel:

1. $a > b \Rightarrow$ Se face atribuirea: $a=18-12=6$.
2. $b > a \Rightarrow$ Se face atribuirea: $b=12-6=6$.
3. $a=b$ ($6=6$) $\Rightarrow \text{cmmdc} = a = 6$.

Varianta 1

```

întreg a,b,r;
început
citește a,b;
cât timp b<>0 execută
    r ← a mod b;
    a ← b;
    b ← r;
sfârșit cât timp;
scrie "cmmdc=",a;
sfârșit.

```

Varianta 2

```

întreg a,b;
început
citește a,b;
cât timp a<>b execută
    dacă a>b
        atunci a ← a-b;
        altfel b ← b-a;
    sfârșit dacă
sfârșit cât timp;
scrie "cmmdc=",a;
sfârșit.

```

Studiu de caz

Scop: exemplificarea modului în care poate fi folosit algoritmul pentru calcularea c.m.m.d.c. a două numere.

Informatică

Enunțul problemei: Să se calculeze cel mai mic multiplu comun și cel mai mare divizor comun a două numere naturale a și b care se introduc de la tastatură.

Calcularea c.m.m.m.c. a două numere se bazează pe calculul c.m.m.d.c. Dacă notăm $\text{cmmdc}(a,b)$ cu c și $\text{cmmmc}(a,b)$ cu m , atunci $a = x \times c$, iar $b = y \times c$, unde x și y sunt prime între ele. Rezultă că $m = x \times y \times c = (a/c) \times (b/c) \times c = a \times b / c$.

Deoarece prin algoritmul de calcul al c.m.m.d.c. se pierd valorile inițiale ale lui a și b , ele se vor salva în două variabile de memorie: x și y .

În cazul în care a și b au ambele valoarea 0, c.m.m.d.c. nu se poate calcula.

Algoritmul este:

```

întreg a,b,c,r,x,y,m;
inceput
  citește a,b;
  x ← a; y ← b;
  dacă b=0
    atunci c ← a;
  altfel
    cât timp b <> 0 execută
      r ← a mod b; a ← b; b ← r;
    sfârșit cât timp;
  c ← a;
sfârșit dacă;
dacă c=0
  atunci scrie "Nu se pot calcula; ambele numere sunt 0";
altfel
  dacă x=0 or y=0
    atunci scrie "Nu se poate calcula c.m.m.m.c.";
    scrie "c.m.m.d.c. = ", c;
  altfel m ← x*y/c;
    scrie c,m;
  sfârșit dacă
sfârșit dacă
sfârșit.

```

Alegeți pentru testarea algoritmului o mulțime completă de seturi de date de intrare. Testați algoritmul.



Probleme care se pot rezolva cu ajutorul algoritmilor pentru calculul c.m.m.d.c.:

1. Se citesc de la tastatură două numere naturale n și k ($2 \leq k \leq n$). Să se afișeze toate perechile de numere naturale mai mici decât n al căror c.m.m.d.c. este k .
2. Să se scrie algoritmul prin care se calculează c.m.m.d.c. și c.m.m.m.c. a 3 numere introduse de la tastatură. Să se generalizeze problema pentru n numere introduse de la tastatură.
3. Să se afișeze toate numerele naturale, mai mici decât un număr natural n , care sunt prime cu n , n introducându-se de la tastatură (două numere naturale se numesc **prime între ele** dacă cel mai mare divizor comun al lor este 1).

3.3.5. Algoritmi pentru testarea unui număr prim

Algoritmul de verificare dacă un număr natural n este prim constă în generarea tuturor numerelor naturale mai mari sau egale cu 2 și mai mici sau egale cu $\sqrt{n}$ și verificarea, pentru fiecare număr generat, dacă îl divide pe n . Dacă există cel puțin un astfel de număr, numărul n nu este prim. Pentru a ști dacă există cel puțin un număr care îl divide pe n , se va folosi o variabilă logică x , care va avea valoarea *True* dacă numărul este prim și *False* dacă numărul nu este prim. Se presupune că numărul este prim (variabila x se inițializează cu valoarea *True*) și, pentru primul număr găsit în șirul de numere generate care îl divide pe n , se va schimba valoarea variabilei x în *False* (numărul nu mai este considerat prim). Pentru generarea șirului de numere se folosește o variabilă contor i care va fi inițializată cu valoarea 2 și care se va incrementa cu 1 până va avea valoarea $\sqrt{n}$. Pașii care se execută sunt:

- Pasul 1.** Se inițializează variabila x prin operația $x \leftarrow T$.
Pasul 2. Se generează prima cifră din șirul de numere, prin operația $i \leftarrow 2$.
Pasul 3. Dacă n se divide cu i , atunci se schimbă valoarea variabilei x prin operația $x \leftarrow F$; altfel, se generează următoarea cifră din șirul de numere, prin incrementarea contorului $i \leftarrow i+1$.
Pasul 4. Dacă $i \leq \sqrt{n}$ (nu s-au generat toate numerele care ar putea fi divizori) și dacă $x = T$ (numărul este considerat în continuare prim), atunci se revine la Pasul 3.
Pasul 5. Dacă $x = T$, se afișează mesajul "Numarul este prim"; altfel, se afișează mesajul "Numarul nu este prim".

```

întreg n,i;
logic x;
început
  citește n;
  x ← T; i ← 2;
  cât timp i ≤ sqrt(n) and x execută
    dacă n mod i = 0
      atunci x ← F;
    altfel i ← i+1;
  sfârșit dacă;
  sfârșit cât timp;
  dacă x
    atunci scrie "Numarul este prim";
    altfel scrie "Numarul nu este prim";
  sfârșit dacă;
sfârșit.

```

Se observă că algoritmul se poate optimiza prin eliminarea, din șirul generat, a numerelor pare, deoarece, dacă numărul n nu se divide prin 2, nu se va divide prin nici un număr par. Algoritmul va fi:

```

întreg n,i;
logic x;
început
  citește n; x ← T;

```

```

dacă  $n \bmod 2 = 0$ 
    atunci  $x \leftarrow F$ ;
    altfel  $i \leftarrow 3$ ;
        cât timp  $i \leq \sqrt{n}$  and  $x$  execută
            dacă  $n \bmod i = 0$ 
                atunci  $x \leftarrow F$ ;
                altfel  $i \leftarrow i+2$ ;
            sfârșit_dacă;
        sfârșit_cât_timp;
    sfârșit_dacă;
dacă  $x$ 
    atunci scrie "Numarul este prim";
    altfel scrie "Numarul nu este prim";
sfârșit_dacă;
sfârșit.

```

Problema se poate rezolva și fără să se folosească variabila x , prin testarea valorii lui i . În cazul în care numărul nu mai este considerat prim, i va lua o valoare în afara șirului de valori generate. Algoritmul va fi:

- Pasul 1.** Se generează prima cifră din șirul de numere, prin operația $i \leftarrow 2$.
- Pasul 2.** Dacă n se divide cu i , atunci se atribuie lui i o valoare în afara șirului de valori generate, $i \leftarrow n$; altfel, se generează următoarea cifră din șirul de numere, prin incrementarea contorului $i \leftarrow i+1$.
- Pasul 3.** Dacă $i \leq \sqrt{n}$, atunci se revine la **Pasul 2**.
- Pasul 4.** Dacă $i > n$ se afișează mesajul "Numarul este prim"; altfel, se afișează mesajul "Numarul nu este prim".

```

Intreg n, i;
Inceput
    citește n;
    dacă  $n \bmod 2 = 0$ 
        atunci  $i \leftarrow n$ ;
        altfel  $i \leftarrow 3$ ;
        cât timp  $i \leq \sqrt{n}$  execută
            dacă  $n \bmod i = 0$ 
                atunci  $i \leftarrow n$ ;
                altfel  $i \leftarrow i+2$ ;
            sfârșit_dacă;
        sfârșit_cât_timp;
    sfârșit_dacă;
    dacă  $i > n$ 
        atunci scrie "Numarul este prim";
        altfel scrie "Numarul nu este prim";
    sfârșit_dacă;
sfârșit.

```

Alegeți, pentru testarea algoritmilor, o mulțime completă de seturi de date de intrare. Testați algoritmi.

Studiu de caz

Scop: exemplificarea modului în care poate fi folosit algoritmul pentru testare dacă un număr natural este prim.

Enunțul problemei: Să se afișeze toate numerele prime din intervalul $[a,b]$, a și b introducându-se de la tastatură.

Numerele din intervalul $[a,b]$ se vor genera în variabila de memorie n . Algoritmul va fi:

- Pasul 1.** Se va verifica mai întâi dacă valorile lui a și b sunt în relația de ordine $a < b$, pentru a reprezenta un interval. Dacă $a > b$ se trece la **Pasul 6**.
- Pasul 2.** Se generează primul număr din interval, cu operația $n \leftarrow a$.
- Pasul 3.** Se verifică dacă numărul n este număr prim. Dacă este număr prim se afișează.
- Pasul 4.** Se generează următorul număr din interval, cu operația $n \leftarrow n+1$.
- Pasul 5.** Se verifică dacă nu s-a ajuns la capătul intervalului: dacă $n < b$ se revine la **Pasul 3**; altfel, se termină algoritmul.
- Pasul 6.** Se afișează mesajul: a , " și ", b , " nu formează un interval".

Se vor folosi următoarele variabile de memorie:

- ✓ **Date de intrare:** a și b pentru limitele intervalului.
- ✓ **Date intermediare:** n pentru numărul care se generează și i pentru generarea șirului de numere cu care se împarte n .
- ✓ **Data de ieșire:** n în cazul în care valoarea sa este un număr prim.

```

întreg a,b,n,i;
început
citește a,b;
dacă a<b
    atunci pentru n ← a,b execută
        dacă n mod 2 = 0
            atunci i ← n;
            altfel i ← 3;
            cât timp i < n div 2 execută
                dacă n mod i = 0
                    atunci i ← n;
                    altfel i ← i+2;
                sfârșit dacă;
            sfârșit cât timp;
        sfârșit dacă;
        dacă i < n
            atunci scrie n;
            sfârșit dacă;
        sfârșit pentru;
    altfel scrie a, " și ", b, " nu formează un interval"
sfârșit.

```



Probleme care se pot rezolva cu ajutorul algoritmului de testare dacă un număr natural este prim:

1. Se introduc n numere de la tastatură. Să se afișeze numerele prime.

2. Să se afișeze primele n numere prime, n introducându-se de la tastatură.
3. Să se afișeze primele n numere prime care au suma cifrelor mai mică decât un număr m , n și m introducându-se de la tastatură.
4. Să se afișeze toate numerele prime de patru cifre care au inversul tot număr prim.
5. Să se afișeze descompunerea unui număr natural par, strict mai mare decât 2, într-o sumă de două numere prime (verificarea ipotezei lui Goldbach).
6. Să se afișeze primele n perechi de **numere prime gemene**, unde n este un număr natural introdus de la tastatură. (Două numere prime a și b sunt gemene dacă $b-a=2$. Exemple: 3 și 5, 5 și 7, 11 și 13, 17 și 19, 29 și 31).
7. Să se afișeze primele n numere naturale strict mai mari decât 2, care au proprietatea că toate numerele naturale strict mai mici decât ele, care sunt prime cu ele, sunt și numere prime (exemplu: $3 \rightarrow 2$; $4 \rightarrow 3$; $6 \rightarrow 5$; contraexemplu: $5 \rightarrow 2, 3, 4$ – deoarece 4 este prim cu 5 dar nu este număr prim).
8. Să se afișeze cel mai mare număr prim, mai mic decât un număr dat n (exemplu: dacă $n=10$, numărul va fi 7).
9. Să se afișeze cel mai mic număr prim, mai mare decât un număr dat n (exemplu: dacă $n=10$, numărul va fi 11).
10. Să se afișeze numerele prime imediat vecine unui număr n (numerele a și b care îndeplinesc condițiile: a și b sunt numere prime, $a \leq n \leq b$ și diferența $b-a$ este minimă).
11. Se citește un număr natural k de la tastatură. Să se afișeze toate numerele n care au k cifre și următoarele proprietăți: a) $n-1$ și $n+1$ sunt numere prime; b) suma cifrelor lui n este tot număr prim. (**Indicație.** Se calculează mai întâi limitele intervalului în care n poate lua valori: cel mai mic număr cu k cifre și cel mai mare număr cu k cifre).

3.3.6. Algoritmi pentru prelucrarea divizorilor unui număr

Algoritmul pentru generarea divizorilor proprii ai unui număr

Algoritmul de generare a **divizorilor proprii** ai unui număr n constă în împărțirea numărului la un șir de numere i , $i \in [2, [n/2]]$. Dacă numărul n se împarte la numărul generat, atunci i este divizor al lui n . Pașii algoritmului sunt:

Pasul 1. Afișează divizorii 1 și n .

Pasul 2. Se inițializează șirul de numere cu care se va împărți n , cu primul divizor posibil, prin operația $i \leftarrow 2$.

Pasul 3. Dacă i îl divide pe n , atunci afișează i .

Pasul 4. Se incrementează cu 1 numărul la care se împarte n , prin $i \leftarrow i+1$.

Pasul 5. Dacă $i \leq [n/2]$, atunci se revine la **Pasul 3**; altfel, se termină algoritmul.

De exemplu, dacă $n=36$, afișarea divizorilor proprii se desfășoară astfel:

1. Afișează 1, 36.
2. Se inițializează împărțitorul cu 2, prin operația de atribuire $i=2$.
3. 2 îl divide pe 36 ($36 \bmod 2 = 0$) $\Rightarrow$ Afișează 2.
4. Se incrementează împărțitorul cu 1: $i=2+1=3$.
5. $i \leq 18$ ($3 \leq 18$) $\Rightarrow$ 3 îl divide pe 36 ($36 \bmod 3 = 0$) $\Rightarrow$ Afișează 3.
6. Se incrementează împărțitorul cu 1: $i=3+1=4$.



... dacă un

7. $i \leq 18$ ($4 \leq 18$) $\Rightarrow$ 4 îl divide pe 36 ($36 \bmod 4 = 0$) $\Rightarrow$ Afișează 4.
8. Se incrementează împărțitorul cu 1: $i = 4 + 1 = 5$.
9. $i \leq 18$ ($5 \leq 18$) $\Rightarrow$ 5 nu îl divide pe 36 ($36 \bmod 5 = 1$).
10. Se incrementează împărțitorul cu 1: $i = 5 + 1 = 6$.
11. $i \leq 18$ ($6 \leq 18$) $\Rightarrow$ 6 îl divide pe 36 ($36 \bmod 6 = 0$) $\Rightarrow$ Afișează 6.
12. Se incrementează împărțitorul cu 1: $i = 6 + 1 = 7$.
-
24. Se incrementează împărțitorul cu 1: $i = 17 + 1 = 18$.
25. $i \leq 18$ ($18 \leq 18$) $\Rightarrow$ 18 îl divide pe 36 ($36 \bmod 18 = 0$) $\Rightarrow$ Afișează 18.
26. Se incrementează împărțitorul cu 1: $i = 18 + 1 = 19$.
27. $i > 18$ ($19 > 18$) $\Rightarrow$ Se termină algoritmul.

Se observă că dacă i este un divizor al lui n , atunci și n/i este un divizor al lui n și căutarea divizorilor se poate limita până la radical de ordinul 2 din n , obținându-se varianta a doua a algoritmului descris prin pseudocod.

Varianta 1

```

întreg n, i;
început
citește n;
scrie 1, n
pentru i ← 2, n div 2 execută
    dacă n mod i = 0
        atunci scrie i;
sfârșit dacă;
sfârșit pentru;
sfârșit.

```

Varianta 2

```

întreg n, i;
început
citește n;
scrie 1, n
pentru i ← 2, [sqrt(n)] execută
    dacă n mod i = 0
        atunci scrie i, n div i;
sfârșit dacă;
sfârșit pentru;
sfârșit.

```

Algoritmul pentru generarea divizorilor primi ai unui număr

Pentru afișarea numai a **divizorilor primi** ai unui număr n , algoritmul anterior se modifică prin eliminarea tuturor divizorilor i găsiți la un moment dat, operația repetându-se până când se elimină toți divizorii din numărul n (n are valoarea 1). Pașii algoritmului sunt:

- Pasul 1.** Se inițializează șirul de numere cu care se va împărți n , cu primul divizor posibil, prin operația $i \leftarrow 2$.
- Pasul 2.** Dacă i îl divide pe n , atunci se afișează i și, atât timp cât n se împarte la i , se împarte n la i pentru a elimina toate puterile lui i din numărul n .
- Pasul 3.** Se trece la următorul divizor posibil, prin incrementarea lui i , cu $i \leftarrow i + 1$.
- Pasul 4.** Dacă $n < 1$, atunci se revine la **Pasul 2**; altfel, se termină algoritmul.

De exemplu, dacă $n = 36$, afișarea divizorilor proprii se desfășoară astfel:

1. Se inițializează împărțitorul cu 2 prin operația de atribuire $i = 2$.
2. 2 îl divide pe 36 ($36 \bmod 2 = 0$) $\Rightarrow$ Afișează 2.
3. 2 îl divide pe 36 $\Rightarrow$ Se împarte n la 2: $n = 36 / 2 = 18$.
4. 2 îl divide pe 18 $\Rightarrow$ Se împarte n la 2: $n = 18 / 2 = 9$.
5. 2 nu îl divide pe 9 $\Rightarrow$ Se incrementează împărțitorul cu 1: $i = 2 + 1 = 3$.
6. $n < 1$ ($9 < 1$) $\Rightarrow$ 3 îl divide pe 9 ($9 \bmod 3 = 0$) $\Rightarrow$ Afișează 3.
7. 3 îl divide pe 9 $\Rightarrow$ Se împarte n la 3: $n = 9 / 3 = 3$.
8. 3 îl divide pe 3 $\Rightarrow$ Se împarte n la 3: $n = 3 / 3 = 1$.
9. 3 nu îl divide pe 1 $\Rightarrow$ Se incrementează împărțitorul cu 1: $i = 3 + 1 = 4$.
10. $n = 1$ ($1 = 1$) $\Rightarrow$ Se termină algoritmul.

```

întreg n,i;
început
citeşte n; i ← 2;
cât timp n<>1 execută
    dacă n mod i = 0
        atunci scrie i;
        cât timp n mod i = 0
            n ← n div i
        sfârşit_cât_timp;
    sfârşit_dacă;
    i ← i+1;
sfârşit_cât_timp;
sfârşit.

```

Studiu de caz

Scop: exemplificarea modului în care poate fi folosit algoritmul pentru generarea divizorilor unui număr.

Enunțul problemei: Să se descompună în factori primi un număr n introdus de la tastatură (se va afișa sub forma a la puterea b , unde a^b este unul dintre factorii primi).

- Pasul 1.** Se inițializează șirul de numere cu care se va împărți n , cu primul divizor posibil, prin operația $i \leftarrow 2$.
- Pasul 2.** Dacă i îl divide pe n , atunci se afișează i , se inițializează contorul k în care se numără puterea divizorului i cu 0, prin operația $k \leftarrow 0$, și atât timp cât n se împarte la i se incrementează contorul k , prin operația $k \leftarrow k+1$, și se împarte n la i pentru a elimina toate puterile lui i din numărul n , după care se afișează k .
- Pasul 3.** Dacă $n \neq 1$, atunci se trece la următorul divizor posibil incrementându-se i , prin operația $i \leftarrow i+1$, și se revine la **Pasul 2**, altfel se termină algoritmul.

```

întreg n,i,k;
început
citeşte n;
i ← 2;
cât timp n<>1 execută
    dacă n mod i = 0
        atunci k ← 0;
        cât timp n mod i = 0
            k ← k+1; n ← n div i;
        sfârşit_cât_timp;
        scrie i, " la puterea ", k;
    sfârşit_dacă;
    i ← i+1;
sfârşit_cât_timp;
sfârşit.

```



Probleme care se pot rezolva cu ajutorul algoritmului de prelucrare a divizorilor unui număr:

1. Să se scrie algoritmul prin care se afișează suma și produsul divizorilor primi ai unui număr natural n care se introduce de la tastatură.
2. Să se scrie algoritmul prin care se determină toate numerele naturale perfecte mai mici decât un număr n introdus de la tastatură. Un număr natural se numește **număr perfect** dacă este egal cu suma divizorilor săi, din care se exclude divizorul egal cu numărul însuși. Exemplu: $6=1+2+3$; $28=1+2+4+7+14$.
3. Să se rezolve, în mulțimea numerelor naturale, ecuația $x^2-y^2=k$, unde k este un număr natural care se introduce de la tastatură. (Indicație: $x^2-y^2=(x+y) \times (x-y)=axb=k \Rightarrow x=(a+b)/2$ și $y=(b-a)/2$; pentru fiecare divizor a al lui k și $b=k/a$ se calculează x și y , soluțiile x și y obținute trebuind să fie numere întregi pozitive.)
4. Se introduce de la tastatură un număr prim p și se citesc pe rând de la tastatură mai multe numere naturale până când se citește numărul 0. Să se determine numărul maxim n astfel încât p^n să dividă produsul numerelor naturale introduse de la tastatură, fără să se calculeze produsul acestor numere.
5. Se citesc n numere naturale diferite de 0. Pentru fiecare număr citit să se afișeze divizorii pari. Dacă nu are divizori pari, să se afișeze un mesaj de informare.

3.3.7. Algoritmi pentru conversii între sisteme de numerație

Algoritmul pentru conversia din baza 10 în baza q

Conversia unui număr n_{10} din baza 10 într-un număr n_q reprezentat în baza q ($2 \leq q \leq 9$) se face prin împărțirea întreagă a numărului la baza q până când restul obținut este mai mic decât baza. Resturile obținute în urma acestor operații de împărțire reprezintă cifrele reprezentării numărului în baza q , primul rest fiind cifra cea mai puțin semnificativă, iar ultimul rest cifra cea mai semnificativă a reprezentării numărului. Pentru a scrie pe ecran numărul obținut în urma conversiei, se va folosi scrierea lui ca un număr în baza 10 (cu cifrele corespunzătoare reprezentării în baza q). Pașii algoritmului sunt:

- Pasul 1.** Se inițializează numărul n_q cu 0, prin operația $n_q \leftarrow 0$.
- Pasul 2.** Se inițializează p – puterea lui 10 – cu 1, prin operația $p \leftarrow 1$.
- Pasul 3.** Se împarte n_{10} la q și se obține restul c , care va fi una dintre cifrele reprezentării ($c \leftarrow n_{10} \bmod q$), și câtul n_{10} ($n_{10} \leftarrow n_{10} \div q$).
- Pasul 4.** Se actualizează formatul de afișare a reprezentării numărului în baza q , prin operația $n_q \leftarrow n_q + c * p$.
- Pasul 5.** Se crește puterea lui 10, prin operația $p \leftarrow p * 10$.
- Pasul 6.** Dacă $n_{10} > 0$, atunci se revine la **Pasul 3**; altfel, se afișează n_q .

De exemplu, dacă $n_{10}=11$ și $q=2$, conversia se desfășoară astfel:

1. $n_q=0$; $p=1$;
2. Se calculează: $c=11 \bmod 2=1$ și $n_{10}=11 \div 2=5$.
3. Se actualizează formatul de afișare, $n_q=0+1*1=1$, și puterea lui 10, $p=1*10=10$.
4. $n_{10}>0$ ($5>0$) $\Rightarrow$ Se calculează: $c=5 \bmod 2=1$ și $n_{10}=5 \div 2=2$.
5. Se actualizează formatul de afișare, $n_q=1+1*10=11$, și puterea lui 10, $p=10*10=100$.
6. $n_{10}>0$ ($2>0$) $\Rightarrow$ Se calculează: $c=2 \bmod 2=0$ și $n_{10}=2 \div 2=1$.

7. Se actualizează formatul de afișare, $nq=11+0*100=11$, și puterea lui 10, $p=100*10=1000$.
8. $n10 < 0$ ($1 > 0$) $\Rightarrow$ Se calculează: $c=1 \bmod 2=1$ și $n10=1 \div 2=0$.
9. Se actualizează formatul de afișare, $nq=11+1*1000=1011$, și $p=1000*10=10000$.
10. $n10=0$ ($0=0$) $\Rightarrow$ Se afișează nq , adică 1011.

Algoritmul pentru conversia din baza q în baza 10

Conversia unui număr nq , din baza q ($2 \leq q \leq 9$), într-un număr $n10$ reprezentat în baza 10, se face folosind descompunerea numărului după puterile bazei:

$$nq = a_n a_{n-1} \dots a_1 a_0 = a_n \times q^n + a_{n-1} \times q^{n-1} + \dots + a_1 \times q^1 + a_0 \times q^0$$

Numărul $n10$ este o sumă în care termenii sunt produsele dintre cifra reprezentării în baza q și puterea corespunzătoare a bazei. Deoarece citirea cifrelor se face începând cu cifra cea mai semnificativă, algoritmul folosește următorul mod de grupare a termenilor reprezentării numărului în baza 10:

$$n10 = ((\dots((a_n \times q + a_{n-1}) \times q + a_{n-2}) \times q + \dots + a_1) \times q + a_0$$

Pașii care se execută sunt:

Pasul 1. Se inițializează numărul $n10$ cu 0, prin operația $n10 \leftarrow 0$.

Pasul 2. Se citește cifra c .

Pasul 3. Se adună, la numărul $n10$ înmulțit cu q , cifra citită, prin operația $n10 \leftarrow n10 * q + c$.

Pasul 4. Se citește cifra c .

Pasul 5. Dacă c este o cifră ($c \geq 0$ și $c < q$), atunci se revine la **Pasul 3**; altfel, se afișează numărul $n10$.

De exemplu, dacă $nq=1011$ și $q=2$, conversia se desfășoară astfel:

1. $n10=0$;
2. Se citește $c=1$
3. $0 \leq c < 2$ ($0 \leq 1 < 2$) $\Rightarrow$ Se calculează: $n10 = 0*2+1 = 1$.
4. Se citește $c=0$
5. $0 \leq c < 2$ ($0 \leq 0 < 2$) $\Rightarrow$ Se calculează: $n10 = 1*2+0 = 2$.
6. Se citește $c=1$
7. $0 \leq c < 2$ ($0 \leq 1 < 2$) $\Rightarrow$ Se calculează: $n10 = 2*2+1 = 5$.
8. Se citește $c=1$
9. $0 \leq c < 2$ ($0 \leq 1 < 2$) $\Rightarrow$ Se calculează: $n10 = 5*2+1 = 11$.
10. Se citește $c=3$
11. $c < 0$ sau $c \geq 2$ ($3 \geq 2$) $\Rightarrow$ Se afișează 11.

Algoritmii sunt:

```
n10  $\Rightarrow$  nq
întreg n10, nq, p, q;
început
  citește n10, q;
  nq  $\leftarrow$  0; p  $\leftarrow$  1;
  cât timp n10 < 0 execută
    nq  $\leftarrow$  nq + p * (n10 mod q);
    n10  $\leftarrow$  n10 div q;
    p  $\leftarrow$  p * 10;
  sfârșit_cât_timp;
  scrie nq;
sfârșit.
```

```
nq  $\Rightarrow$  n10
întreg c, n10, q;
început
  citește q;
  n10  $\leftarrow$  0;
  citește c;
  cât timp c >= 0 and c < q execută
    n10  $\leftarrow$  n10 * q + c;
    citește c;
  sfârșit_cât_timp
  scrie n10;
sfârșit.
```


Probleme care se pot rezolva cu ajutorul algoritmilor de conversie între sisteme de numerație

1. Se citește un număr natural n de la tastatură. Să se afișeze reprezentarea lui în baza q , $q \in [2, 9]$. q și n se introduc de la tastatură.
2. Se citesc de la tastatură un număr natural n și un număr q , $q \in [2, 9]$. Să se verifice dacă n poate fi considerat ca o reprezentare a unui număr în baza q .
3. Se citește un număr natural n de la tastatură. Să se afișeze câte cifre are reprezentarea lui în baza q , $q \in [2, 9]$. q și n se introduc de la tastatură.
4. Se citesc de la tastatură q , baza de numerație, $q \in [2, 9]$, și mai multe numere naturale care reprezintă cifrele unui număr în baza q , până când numărul introdus nu mai poate fi considerat cifră în această bază de numerație. Să se afișeze numărul reprezentat în baza 10.
5. Se citește de la tastatură un număr n , care este reprezentarea numărului în baza q , $q \in [2, 9]$. Să se afișeze numărul reprezentat în baza 10. q se introduce de la tastatură.
6. Se citește de la tastatură un număr n , care este reprezentarea numărului în baza q , $q \in [2, 9]$. Să se afișeze câte cifre are reprezentarea lui în baza 10. q se introduce de la tastatură.
7. Se citește de la tastatură un număr n , care este reprezentarea numărului în baza q , $q \in [2, 9]$. Să se afișeze numărul reprezentat în baza p , $p \in [2, 9]$. q și p se introduc de la tastatură.
8. Se citește un număr în baza 4. Să se verifice că numărul este corect (cifrele sale corespund bazei de numerație) și să se afișeze frecvența cifrelor sale.
9. Se introduc de la tastatură baza de numerație k , $k \in [2, 9]$, și cifrele unui număr reprezentat în baza k . Să se verifice dacă cifrele sunt corecte pentru baza de numerație aleasă și să se afișeze suma cifrelor de rang par (impar). Citirea cifrelor se face începând cu cifra cea mai semnificativă a numărului.
10. Să se genereze toate numerele naturale a căror reprezentare în baza 8 are exact 4 cifre.
11. Să se genereze toate numerele naturale a căror reprezentare în baza q are exact k cifre; q și k se introduc de la tastatură și $q \in [2, 9]$.
12. Să se afișeze toate numerele naturale mai mici decât un număr dat n , care se introduce de la tastatură ($n \geq 7$), a căror reprezentare în baza 2 conține exact trei cifre binare de 1.
13. Să se afișeze toate numerele naturale cuprinse între a și b ($a < b$) care au proprietatea că pătratul și cubul lor, reprezentate în baza 2, conțin același număr de cifre binare 0. a și b se introduc de la tastatură.
14. Să se afișeze toate numerele în baza 2 care au k cifre și sunt divizibile cu 2^n . k și n se introduc de la tastatură. (Indicație. Un număr reprezentat în baza 2 este divizibil cu 2^n dacă ultimele n cifre ale reprezentării sunt 0).
15. Să se afișeze toate numerele naturale mai mici decât n care sunt palindrom în baza k . n și k se introduc de la tastatură și $k \in [2, 9]$.

16. Să se afișeze toate numerele reprezentate în baza k , mai mici decât n , care sunt palindrom în baza 10. n și k se introduc de la tastatură $k \in [2, 9]$.

3.3.8. Algoritmi pentru generarea șirurilor recurente

Algoritmul pentru generarea termenilor șirului lui Fibonacci este un exemplu clasic de algoritmi pentru **definirea recurentă** a termenilor unui șir. Șirul lui Fibonacci este format din termeni definiți prin recurență, astfel:

$$\begin{aligned} a_1 &= 1 \\ a_2 &= 1 \\ a_3 &= a_1 + a_2 \\ &\dots \\ a_n &= a_{n-2} + a_{n-1} \end{aligned}$$

Pentru generarea primilor n termeni ai șirului lui Fibonacci ($n \geq 3$), se vor genera repetat termenii de rang i ai șirului, cu $3 \leq i \leq n$. Se vor folosi trei variabile de memorie: a_1 pentru a_{i-2} , a_2 pentru a_{i-1} și a_3 pentru termenul curent a_i . După fiecare generare a unui termen se vor actualiza valorile din variabilele a_1 și a_2 . Pașii algoritmului sunt:

Pasul 1. Se inițializează termenii a_1 și a_2 , prin atribuirile $a_1 \leftarrow 1$ și $a_2 \leftarrow 1$.

Pasul 2. Se afișează termenii a_1 și a_2 .

Pasul 3. Se inițializează contorul i care numără termenii generați, prin operația $i \leftarrow 3$ (urmează să se calculeze termenul 3).

Pasul 4. Se calculează a_3 , prin operația de atribuire $a_3 \leftarrow a_1 + a_2$.

Pasul 5. Se afișează termenul a_3 .

Pasul 6. Se actualizează valorile pentru a_1 și a_2 , prin atribuirile $a_1 \leftarrow a_2$ și $a_2 \leftarrow a_3$.

Pasul 7. Se incrementează contorul cu 1, prin operația $i \leftarrow i + 1$.

Pasul 8. Dacă $i \leq n$, atunci se revine la **Pasul 4**; altfel, se termină algoritmul.

De exemplu, dacă $n=5$, generarea termenilor se desfășoară astfel:

1. $a_1=1$; $a_2=1$; Afișează 1, 1. Se inițializează contorul $i=3$.
2. $a_3=1+1=2$. Afișează 2.
3. Se actualizează termenii a_1 și a_2 : $a_1=a_2=1$ și $a_2=a_3=2$.
4. Se incrementează contorul cu 1: $i=3+1=4$.
5. $i \leq 5$ ($4 \leq 5$) $\Rightarrow$ Se calculează $a_3=1+2=3$. Se afișează 3.
6. Se actualizează termenii a_1 și a_2 : $a_1=a_2=2$ și $a_2=a_3=3$.
7. Se incrementează contorul cu 1: $i=4+1=5$.
8. $i \leq 5$ ($5 \leq 5$) $\Rightarrow$ Se calculează $a_3=2+3=5$. Se afișează 5.
9. Se actualizează termenii a_1 și a_2 : $a_1=a_2=3$ și $a_2=a_3=5$.
10. Se incrementează contorul cu 1: $i=5+1=6$.
11. $i > 5$ ($6 > 5$) $\Rightarrow$ Se termină algoritmul.

```

întreg a1,a2,a3,n,i;
început
  citește n;
  a1 ← 1; a2 ← 1;
  scrie a1,a2
  pentru i←3,n execută
    a3 ← a1+a2; scrie a3;
    a1 ← a2; a2 ← a3;
  sfârșit pentru;
sfârșit.

```


Probleme care se pot rezolva cu ajutorul algoritmilor de generare a șirurilor recurente

1. Să se afișeze toți termenii șirului lui Fibonacci mai mici decât un număr natural n introdus de la tastatură.
2. Să se determine dacă un număr n introdus de la tastatură poate fi un termen al șirului lui Fibonacci.
3. Să se verifice dacă două numere naturale n și m introduse de la tastatură ($m \geq n$) pot fi termeni consecutivi ai șirului lui Fibonacci, fără a se calcula termenii șirului. (Indicație. Se execută operația inversă, de determinare a termenilor precedenți: inițializarea, cu $a_3 \leftarrow m$, $a_2 \leftarrow n$, $a_1 \leftarrow m - n$, și generarea, cu $a_3 \leftarrow a_2$, $a_2 \leftarrow a_1$ și $a_1 \leftarrow a_3 - a_2$, cât timp $a_1 > 0$; dacă $a_3 = a_2 = 1$, m și n sunt termeni consecutivi ai șirului lui Fibonacci.)
4. Fiind date x un număr real și n un număr natural ($n \geq 3$), care se introduc de la tastatură, să se afișeze $P_n(x)$, definit recurent astfel:

$$P_1(x) = x$$

$$P_2(x) = x - 2$$

$$P_3(x) = x P_2(x) - P_1(x)$$

$$\dots\dots\dots$$

$$P_n(x) = x P_{n-1}(x) - P_{n-2}(x)$$
5. Se citesc trei numere întregi a , b și c care reprezintă coeficienții unei ecuații de gradul 2, și un număr natural n . Să se calculeze $S_n = x_1^n + x_2^n$, unde x_1 și x_2 sunt rădăcinile ecuației. Suma se calculează fără a se rezolva ecuația de gradul 2. Notăm cu S suma rădăcinilor ($S = -b/a$) și cu P produsul rădăcinilor ($P = c/a$). Atunci: $S_n = S(x_1^{n-1} + x_2^{n-1}) - P(x_1^{n-2} + x_2^{n-2}) = S \times S_{n-1} - P \times S_{n-2}$. Știind că $S_0 = 1 + 1 = 2$ și $S_1 = S$, se va folosi definiția recurentă:

$$S_0 = 2$$

$$S_1 = S$$

$$S_2 = S \times S_1 - P \times S_0$$

$$\dots\dots\dots$$

$$S_n = S \times S_{n-1} - P \times S_{n-2}$$
6. Să se calculeze rădăcina pătrată dintr-un număr real x , prin generarea unui șir de numere $(a_i)_{i \leq n}$ care o aproximează, folosind definiția recurentă:

$$a_1 = 1$$

$$a_2 = (1 + x/1)/2$$

$$a_3 = ((1 + x)/2 + x/((1 + x)/2))/2$$

$$\dots\dots\dots$$

$$a_n = (a_{n-1} + x/a_{n-1})/2$$

Termenii a_i vor fi generați recursiv până când diferența $|a_i - a_{i-1}|$ va fi mai mică decât o valoare ϵ (eroarea acceptată, un număr subunitar foarte mic). Ultimul termen a_i generat va conține valoarea aproximativă a radicalului din numărul x . x și ϵ se introduc de la tastatură.
7. Să se afișeze primii n termeni ai șirului (n se introduce de la tastatură): 1, 1, 2, 1, 2, 3, 1, 2, 3, 4, 1, 2, 3, 4, 5, ...
8. Să se afișeze primii n termeni ai șirului (n se introduce de la tastatură): 1, 2, 1, 1, 2, 3, 4, 3, 2, 1, 1, 2, 3, 4, 5, 6, 5, 4, 3, 2, 1, ...

3.4. Eficiența algoritmilor

Ați văzut că pentru rezolvarea unei probleme se pot folosi mai mulți algoritmi. În acest caz se va alege **algoritmul cel mai eficient**.

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

- ✓ **Memoria internă.** În memoria internă se alocă spațiu atât pentru datele folosite de algoritm, cât și pentru codul executabil al programului (instrucțiunile în cod mașină).
- ✓ **Procesorul.** Timpul de utilizare a procesorului depinde de timpul necesar pentru executarea algoritmului.

Memorie internă

Din punct de vedere al gândirii algoritmului, pentru a face economie de această resursă, trebuie avute în vedere următoarele:

- ✓ Alegerea corectă a tipului de dată pentru fiecare variabilă de memorie folosită în algoritm.
- ✓ Rezolvarea problemei folosind cât mai puține variabile de memorie.

Ați văzut că atunci când i se atribuie unei date un tip de dată, data capătă mai multe atribute care determină **domeniul de definiție intern** al ei (mulțimea în care poate lua valori data). Tipul de dată ales influențează calitatea programului, deoarece el determină dimensiunea zonei de memorie alocată, algoritmul de codificare și operații admiși pentru prelucrare. Din această cauză, la alegerea tipului de dată trebuie să se facă în două moduri **analiza datei**:

- ✓ **Logic** (la nivelul conceptual). Analiza se face pornind de la enunțul problemei și constă în identificarea **domeniului de definiție extern** al datei. De exemplu, în enunțul problemei se precizează că trebuie prelucrat un număr întreg pozitiv, cu valori cuprinse între 0 și 200. Acesta este domeniul de definiție extern al datei.
- ✓ **Fizic** (la nivelul reprezentării ei în memoria internă). Analiza se face pornind de la tipurile de date implementate în limbajul de programare, fiecare tip de dată având un **domeniu de definiție intern** al datei. Să presupunem că în limbajul de programare sunt implementate următoarele trei tipuri de date întregi: *tipul 1* cu domeniul de definiție $[-128, 127]$ reprezentat pe un octet, *tipul 2* cu domeniul de definiție $[0, 255]$ reprezentat pe un octet și *tipul 3* cu domeniul de definiție $[-32768, 32767]$ reprezentat pe doi octeți. În urma analizei fizice trebuie ales tipul de dată adecvat. **Regula este: se alege tipul de dată care consumă cea mai puțină memorie, astfel încât domeniul de definiție extern al datei să fie inclus în domeniul de definiție intern al datei.** Pentru exemplul prezentat se va alege *tipul 2* de dată deoarece numai *tipul 2* și *tipul 3* respectă condiția de incluziune a domeniilor de definiție ($[0, 200] \subset [0, 255]$ și $[0, 200] \subset [-32768, 32767]$, dar $[0, 200] \not\subset [-128, 127]$), iar *tipul 2* ocupă mai puțin spațiu de memorie (1 octet) decât *tipul 3* (2 octeți).

Problema pare neimportantă atunci când algoritmul folosește câteva variabile de memorie. Dar trebuie să vă gândiți că pentru rezolvarea problemelor complexe se pot folosi structuri de date în care se memorează foarte multe date elementare (de

la câteva zeci până la milioane de date elementare). În acest caz este important dacă tipul de dată elementară este ales corect. Fiecare octet de care nu are nevoie o dată elementară poate să însemne un consum inutil de milioane de octeți pentru structura de date.

Procesorul

Este evident că timpul de execuție al unui algoritm depinde de câte valori ale datelor de intrare vor fi prelucrate. Într-o formulare de genul „se prelucrează n valori numerice citite de la tastatură” sau într-o formulare de genul „se prelucrează mai multe numere întregi citite de la tastatură, până la întâlnirea valorii 0”, la o execuție a algoritmului se pot prelucra numai 2 valori pentru data de intrare (dacă n are valoarea 2, sau dacă se citesc numai două numere întregi diferite de 0), sau se pot prelucra 100 de valori pentru data de intrare (dacă n are valoarea 100, sau dacă se citesc 100 numere întregi diferite de 0). Se definește **dimensiunea datelor de intrare** ca fiind numărul de valori pentru datele de intrare ale unui algoritm. Pentru compararea timpului de execuție a doi algoritmi care rezolvă aceeași problemă, se va folosi aceeași dimensiune a datelor de intrare.

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

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

O **operație elementară** este o operație sau o succesiune de operații care nu depind de caracteristicile problemei. De exemplu, poate fi operație elementară o operație aritmetică (adunare, scădere etc.), o operație de comparație, o operație de atribuire sau un grup bine precizat de astfel de operații (5 operații de adunare, 2 operații de comparare, 10 operații de atribuire etc.). În schimb, n operații de atribuire nu reprezintă o operație elementară, deoarece depind de o caracteristică a problemei, și anume de valoarea lui n . Există cazuri în care nu se poate preciza numărul de operații elementare, acestea depinzând de valoarea datelor de intrare. Să comparăm următorii doi algoritmi:

Enunțul problemei 1. Se citesc n numere. Să se calculeze suma lor.

Enunțul problemei 2. Se citesc n numere. Să se calculeze suma numerelor pare.

Problema 1

```

întreg n,a,i,s;
început
citește n; s ← 0;
pentru i ← 1,n execută
    citește a;
    s ← s+a;
sfârșit_pentru;
scrie s;
sfârșit.

```

Problema 2

```

întreg n,a,i,s;
început
citește n; s ← 0;
pentru i ← 1,n execută
    citește a;
    dacă a mod 2=0
        atunci s ← s+a;
    sfârșit_dacă;
sfârșit_pentru;
scrie s;
sfârșit.

```

În cazul ambilor algoritmi se execută, în afara structurii repetitive, patru operații elementare (o operație de citire, o atribuire, o inițializare contor și o operație de scriere).

În cazul primului algoritm se execută în structura repetitivă patru operații elementare (două operații asupra contorului i – o comparare și o incrementare –, o operație de citire și o atribuire). În total, se execută $4+4 \times n$ operații.

În cazul celui de al doilea algoritm se execută în structura repetitivă două operații asupra contorului i – o comparare și o incrementare –, o operație de citire, o operație de comparare și nu se poate preciza dacă se execută și atribuirea). În total se execută $4+4 \times n + x$ operații, unde x depinde de câte numere pare se citesc de la tastatură.

În cel de al doilea caz, pentru compararea algoritmilor se folosește **timpul mediu de execuție** al algoritmului. Există un **timp minim de execuție** care corespunde cazului cel mai favorabil (cazul în care se execută cele mai puține operații) și un **timp maxim de execuție** care corespunde cazului cel mai defavorabil (cazul în care se execută cele mai multe operații). Pentru exemplul prezentat cazul cel mai favorabil este ca să nu existe nici un număr par. În acest caz $x=0$ și numărul de operații elementare este $4+4 \times n$. Cazul cel mai defavorabil este ca toate numerele să fie pare. În acest caz $x=n$ și numărul de operații elementare este $4+5 \times n$. Pentru a calcula timpul mediu de execuție, va trebui să analizăm toate cazurile posibile (în total $n+1$ cazuri): nici un număr par, un număr par, două numere pare, ..., n numere pare, numărul mediu de operații fiind dat de x calculat astfel:

$$x = (0+1+2+\dots+n)/(n+1) = (n \times (n+1)/2)/(n+1) = n/2$$

Studiu de caz

Scop: exemplificarea modului în care pot fi comparați doi algoritmi folosind numărul de operații elementare.

Enunțul problemei 1: Să se determine valoarea unui polinom de gradul n . Gradul polinomului n , coeficienții a_i ai polinomului și valoarea lui x se introduc de la tastatură.

Se vor folosi următoarele variabile de memorie;

- ✓ **Date de intrare:** n pentru gradul polinomului, x pentru valoarea pentru care se efectuează calculul și a pentru citirea unui coeficient.
- ✓ **Data intermediară:** i de tip contor, pentru numărarea celor $n+1$ coeficienți citați.
- ✓ **Data de ieșire:** p , pentru valoarea polinomului.

Pentru rezolvarea acestei probleme se vor folosi două variante de algoritmi;

Varianța 1.

$$P(x) = a_n x^n + a_{n-1} x^{n-1} + \dots + a_1 x^1 + a_0 x^0$$

Introducerea coeficienților începe cu coeficientul a_0 și se termină cu coeficientul a_n .

Varianța 2.

$$P(x) = ((\dots((a_n x + a_{n-1}) x + a_{n-2}) x + \dots + a_1) x + a_0$$

Introducerea coeficienților începe cu coeficientul a_n și se termină cu coeficientul a_0 .

Algoritmii sunt:

Varianta 1

```

întreg n,p,x,a,i;
început
  citește n,x,a;
  p ← a;
  pentru i←1,n execută
    citește a;
    p ← p+a*x;
    x ← x*x;
  sfârșit_pentru;
  scrie p;
sfârșit.

```

Varianta 2

```

întreg n,p,x,a,i;
început
  citește n,x,a;
  p ← a;
  pentru i←1,n execută
    citește a;
    p ← p*x+a;
  sfârșit_pentru;
  scrie p;
sfârșit.

```

Se observă că cei doi algoritmi diferă din punct de vedere al structurii repetitive, în care se execută, în prima variantă, un grup de cinci operații (două operații asupra contorului i – o comparare și o incrementare –, o citire și două atribuiri), iar în a doua variantă, un grup de patru operații (două operații asupra contorului i – o comparare și o incrementare –, o citire și o atribuire), în afara structurii repetitive executându-se în ambele cazuri patru operații elementare (o operație de citire a trei date de intrare, o operație de atribuire, o operație de inițializare a contorului și o operație de scriere). Așadar, pentru aceeași dimensiune a datelor de intrare (n), în prima variantă se execută $4+5 \times n$ operații elementare, iar în a doua variantă $4+4 \times n$ operații elementare. Este evident că a doua variantă este cea eficientă.

Enunțul problemei 2: Se introduc n numere de la tastatură. Să se numere câte sunt divizibile cu 2, câte sunt divizibile cu 3 și câte sunt divizibile cu 6.

Se vor folosi următoarele variabile de memorie;

- ✓ **Date de intrare:** n pentru numărul de elemente și a pentru citirea unui număr.
- ✓ **Data intermediară:** i , de tip contor, pentru numărarea celor n numere care se citesc.
- ✓ **Data de ieșire:** k_1 , pentru numărarea numerelor divizibile cu 2, k_2 pentru numărarea numerelor divizibile cu 3 și k_3 pentru numărarea numerelor divizibile cu 6.

Pentru rezolvarea acestei probleme se vor folosi două variante de algoritmi:

Varianta 1

```

întreg n,a,k1,k2,k3,i;
început
  citește n,a;
  k1 ← 0; k2 ← 0; k3 ← 0;
  pentru i←1,n execută
    citește a;
    dacă a mod 2 = 0
      atunci
        k1 ← k1+1;
    dacă a mod 3 = 0
      atunci

```

Varianta 2

```

întreg n,a,k1,k2,k3,i;
început
  citește n,a;
  k1 ← 0; k2 ← 0; k3 ← 0;
  pentru i←1,n execută
    citește a;
    dacă a mod 2 = 0
      atunci
        k1 ← k1+1;
    sfârșit_dacă;
    dacă a mod 3 = 0

```

```

    k2 ← k2+1;
    k3 ← k3+1;
    sfârșit_dacă;
    altfel
    dacă a mod 3 = 0
    atunci
        k2 ← k2+1;
    sfârșit_dacă;
    sfârșit_dacă;
    sfârșit_pentru;
    scrie k1,k2,k3;
sfârșit.

```

```

    atunci
        k2 ← k2+1;
    sfârșit_dacă;
    dacă a mod 6 = 0
    atunci
        k3 ← k3+1;
    sfârșit_dacă;
    sfârșit_dacă;
    sfârșit_pentru;
    scrie k1,k2,k3;
sfârșit.

```

În acest caz, nu se poate preciza numărul de operații elementare ale fiecărui algoritm, deoarece el depinde de fiecare număr care se citește: dacă nu este divizibil nici cu 2 și nici cu 3, dacă este divizibil numai cu 2, dacă este divizibil numai cu 3 sau dacă este divizibil cu 6. Pentru compararea celor doi algoritmi, vom calcula numărul de operații pentru fiecare caz de divizibilitate:

	Varianta 1				Varianta 2			
	nu	2	3	6	nu	2	3	6
comparații	2	2	2	2	3	3	3	3
atribuiri	0	1	1	3	0	1	1	3
total	2	3	3	5	3	4	4	6

Se observă că prima variantă este mai eficientă decât cea de a doua.



Studiu de caz

Scop: exemplificarea modului în care pot fi comparați doi algoritmi folosind timpii de execuție.

Vom considera primele două variante de algoritmi pentru testarea unui număr prim.

În cele două variante se execută sigur:

- ✓ **Varianta 1:** 5 operații elementare: o operație de citire, două atribuiri, o comparație și o operație de scriere.
- ✓ **Varianta 2:** 5 operații elementare: o operație de citire, o atribuire, două comparații și o operație de scriere.

Cazul cel mai favorabil este ca numărul să fie par. În acest caz, se mai execută:

- ✓ **Varianta 1:** 3 operații elementare: două operații de comparație (una în structura repetitivă și una în structura alternativă) și o operație de atribuire – în total 8 operații.
- ✓ **Varianta 2:** o operație de atribuire – în total 6 operații.

Cazul cel mai defavorabil este ca numărul să fie prim. În acest caz se execută:

- ✓ **Varianta 1:** de $\lfloor \sqrt{n} \rfloor - 1$ ori trei operații (două comparații și o atribuire), în total $3 \times (\lfloor \sqrt{n} \rfloor - 1)$ operații – în total $5 + 3 \times (\lfloor \sqrt{n} \rfloor - 1)$ operații.

- ✓ **Varianta 2:** de $(\lceil \sqrt{n} \rceil - 2)/2$ ori trei operații (două comparații și o atribuire), în total $3 \times ((\lceil \sqrt{n} \rceil - 2)/2)$ operații – în total $5 + 3 \times ((\lceil \sqrt{n} \rceil - 2)/2)$ operații.

Este evident că varianta 2 este mai eficientă.



3.5. Aplicarea algoritmilor

Puteți să rezolvați multe dintre problemele de la disciplinele școlare (matematică, fizică sau chimie) cu ajutorul calculatorului. Pentru aceasta trebuie mai întâi să descrieți rezolvarea lor cu ajutorul algoritmilor.

3.5.1. Rezolvarea problemelor de matematică

Enunțul problemei 1: Se citesc trei numere întregi a , b și c care reprezintă coeficienții unei ecuații de gradul 2. Să se rezolve ecuația.

Se vor folosi următoarele variabile de memorie:

- ✓ **Date de intrare:** a , b și c pentru coeficienții ecuației.
- ✓ **Data intermediară:** d pentru discriminant.
- ✓ **Date de ieșire:** x_1 și x_2 pentru rădăcinile reale ale ecuației.

Algoritmul este:

```

real a,b,c,x1,x2,d;
inceput
  citește a,b,c;
  dacă a=0
    atunci
      scrie "Ecuație de gradul întâi";
      dacă b=0
        atunci
          dacă c=0
            atunci scrie " cu o infinitate de solutii";
            altfel scrie " care nu are solutii";
          sfârșit_dacă; // pentru c=0
        altfel
          x1 ← -c/b; scrie " cu radacina", x1;
          sfârșit_dacă; // pentru b=0
      altfel
        d ← b*b-4*a*c;
        dacă d>0
          atunci
            x1 ← (-b+sqrt(d))/(2*a); x2 ← (-b-sqrt(d))/(2*a);
            scrie "Ecuația are doua radacini reale diferite ";
            scrie "x1= ", x1; scrie "x2= ", x2;
          altfel
            dacă d=0
              atunci

```

tribuire), în



matematică,
mai întâi să

ezintă coefici-

```

x1 ← -b/(2*a);
scrie "Ecuatia are doua solutii reale identice";
scrie "x1=x2= ", x1;
altfel
scrie "Ecuatia nu are solutii reale ";
sfârșit_dacă;
sfârșit_dacă;
sfârșit_dacă;
sfârșit.

```

Pentru testarea algoritmului folosiți un set de date de intrare care este format din coeficienții ecuației (a,b,c), iar o mulțime completă de seturi de date de intrare poate fi {(0,0,0), (0,0,1), (0,2,5), (1,-5,6), (1,-2,1), (1,1,1)}. Testați algoritmul.

Enunțul problemei 2: Să se rezolve un sistem de două ecuații liniare cu două necunoscute:

$$\begin{aligned} a_1 \times x + b_1 \times y &= c_1 \\ a_2 \times x + b_2 \times y &= c_2 \end{aligned}$$

Soluțiile sistemului de ecuații sunt:

$$\begin{aligned} x &= d_x/d_s = (b_2 \times c_1 - b_1 \times c_2)/(a_1 \times b_2 - b_1 \times a_2) \\ y &= d_y/d_s = (a_1 \times c_2 - a_2 \times c_1)/(a_1 \times b_2 - b_1 \times a_2) \end{aligned}$$

Se vor folosi următoarele variabile de memorie:

- ✓ **Date de intrare:** a1, b1, c1, a2, b2 și c2 pentru coeficienții sistemului de ecuații.
- ✓ **Data intermediară:** ds, dx și dy pentru d_s, d_x și d_y.
- ✓ **Date de ieșire:** x și y pentru soluțiile ecuației.

Algoritmul este:

```

intreg a1, b1, c1, a2, b2, c2, ds, dx, dy;
real x, y;
inceput
citește a1, b1, c1, a2, b2, c2;
ds ← a1*b2 - b1*a2; dx ← b2*c1 - b1*c2; dy ← a1*c2 - a2*c1;
dacă ds=0
atunci dacă dx=0
atunci scrie "Sistem nedeterminat";
altfel scrie "Sistem incompatibil";
sfârșit_dacă;
altfel
x ← dx/ds; y ← dy/ds;
scrie "x= ", x; scrie "y= ", y;
sfârșit_dacă;
sfârșit.

```

Enunțul problemei 3: Se consideră două segmente de dreaptă în plan, neparalele cu axa Oy, reprezentate prin coordonatele lor. Să se determine intersecția celor două segmente, dacă există, iar în caz contrar, să se specifice cazurile de excepție.

Coordonatele extremităților segmentelor sunt (x₁, y₁) și (x₂, y₂) pentru primul segment și (x₃, y₃) și (x₄, y₄) pentru al doilea segment. Coordonatele punctului de inter-

secție sunt (x_i, y_i) . Panta și ordonata la origine ale dreptelor sunt m_1 și n_1 pentru prima dreaptă și m_2 și n_2 pentru a doua dreaptă.

Se vor folosi următoarele variabile de memorie;

- ✓ **Date de intrare:** $x_1, y_1, x_2, y_2, x_3, y_3, x_4, y_4$ pentru coordonatele extremităților segmentelor.
- ✓ **Data intermediară:** m_1 și m_2 pentru pantele segmentelor, n_1 și n_2 pentru ordonatele la origine ale segmentelor, min_1 și max_1 pentru minimul și maximum dintre abscisele primului segment, min_2 și max_2 pentru minimul și maximum dintre ordonatele primului segment, min_3 și max_3 pentru minimul și maximum dintre abscisele celui de al doilea segment, min_4 și max_4 pentru minimul și maximum dintre ordonatele celui de al doilea segment.
- ✓ **Date de ieșire:** x și y pentru coordonatele punctului de intersecție.

Pasul 1. Se citesc coordonatele extremităților segmentelor.

Pasul 2. Dacă segmentele sunt paralele cu axa Oy ($x_1=x_2$ or $x_3=x_4$) scrie mesajul "Cel puțin o dreaptă este paralelă cu Oy " și termină algoritmul.

Pasul 3. Se calculează pantele m_1 și m_2 .

Pasul 4. Dacă pantele sunt egale ($m_1 = m_2$), treci la **Pasul 5**, altfel treci la **Pasul 6**.

Pasul 5. Dacă un punct de pe prima dreaptă se găsește pe a doua dreaptă (coordo-natele lui verifică ecuația celei de a doua drepte), atunci scrie mesajul "Segmentele sunt pe aceeași dreaptă", altfel scrie mesajul "Segmentele sunt paralele". Termină algoritmul.

Pasul 6. Se calculează coordonatele punctului de intersecție al dreptelor.

Pasul 7. Dacă abscisa punctului de intersecție nu este între abscisele capetelor unuia dintre segmente sau dacă ordonata punctului de intersecție nu este între ordonatele capetelor unuia dintre segmente, atunci scrie mesajul "Segmentele se intersectează pe prelungiri".

Pasul 7. Scrie coordonatele punctului de intersecție.

Algoritmul este:

```

real x1,y1,x2,y2,x3,y3,x4,y4,x,y,m1,n1,m2,n2,min1,max1,
    min2,max2,min3,max3,min4,max4;
început
citește x1,y1,x2,y2,x3,y3,x4,y4;
dacă (x1=x2) or (x3=x4)
    atunci scrie "Cel puțin o dreaptă este paralela cu Oy";
    altfel
        m1 ← (y2-y1)/(x2-x1); m2 ← (y4-y3)/(x4-x3);
        n1 ← y1-m1*x1; n2 ← y3-m2*x3;
        dacă m1=m2
            atunci
                dacă y1=m2*x1+n2
                    atunci scrie "Segmentele sunt pe aceeași dreapta";
                    altfel scrie "Segmentele sunt paralele";
                sfârșit_dacă;
            altfel
                x ← (n2-n1)/(m1-m2); y ← m1*x+n1;
                dacă x1<x2

```

```

        atunci
            min1 ← x1; max1 ← x2;
        altfel
            min1 ← x2; max1 ← x1;
    sfârșit_dacă;
    dacă y1 < y2
        atunci
            min2 ← y1; max2 ← y2;
        altfel
            min2 ← y2; max2 ← y1;
    sfârșit_dacă;
    dacă x3 < x4
        atunci
            min3 ← x3; max3 ← x4;
        altfel
            min3 ← x4; max3 ← x3;
    sfârșit_dacă;
    dacă y3 < y4
        atunci
            min4 ← y3; max4 ← y4;
        altfel
            min4 ← y4; max4 ← y3;
    sfârșit_dacă;
    dacă x < min1 or y < min2 or x < min3 or y < min4 or x > max1
        or y > max2 or x > max3 or y > max4
        atunci scrie "Segmentele se intersectează pe
            prelungiri";
    sfârșit_dacă;
    scrie "x= ", x; scrie "y= ", y;
sfârșit_dacă;
sfârșit_dacă;
sfârșit.

```

Alegeți o mulțime completă de seturi de date de intrare și testați algoritmul.

Probleme de matematică propuse:

1. Se citesc două numere care reprezintă dimensiunea înălțimii și a razei unui cilindru circular drept. Să se calculeze aria laterală, aria totală și volumul cilindrului și ale conului circular drept de aceeași rază și înălțime cu cilindrul.
2. Se citesc trei numere întregi a , b și c . Dacă pot reprezenta laturile unui triunghi, să se calculeze dimensiunile razei cercului înscris, razei cercului circumscris, razelor cercurilor exînscrise și a înălțimilor.
3. Să se determine rădăcinile ecuației $ax^4 + bx^2 + c = 0$. Coeficienții ecuației se citesc de la tastatură.
4. Se citesc de la tastatură numărătorul a și numitorul b ale unei fracții. Să se afișeze fracția simplificată.
5. Se citesc coordonatele unui punct din plan. Să se precizeze unde este situat punctul în plan (numele axei sau numărul cadranelui).

6. Se citesc coordonatele a două puncte din plan. Să se calculeze distanța dintre ele și coordonatele mijlocului segmentului de dreaptă care le unește.
7. Se citesc coordonatele a trei puncte din plan. Să se precizeze dacă punctele sunt coliniare sau nu.
8. Se citesc coeficienții ecuației carteziene generale a unei drepte. Să se precizeze cum este dreapta: oarecare, prima bisectoare, a doua bisectoare, trece prin origine, paralelă cu Ox, paralelă cu Oy, axa Ox sau axa Oy.
9. Se citesc coordonatele vârfurilor unui triunghi. Să se determine coordonatele ortocentrului.
10. Se citesc coordonatele a două vârfuri ale unui triunghi și coordonatele ortocentrului. Să se determine coordonatele celuilalt vârf.
11. Se citesc coeficienții ecuațiilor carteziene generale pentru trei drepte. Să se determine dacă dreptele sunt paralele și echidistante.
12. Se citesc coordonatele vârfului unui pătrat și coeficienții ecuației carteziene generale a unei drepte pe care se găsește una dintre laturile pătratului care nu trece prin acel vârf. Să se determine coordonatele celorlalte vârfuri ale pătratului și aria pătratului.
13. Se citesc coordonatele a trei puncte din plan. Să se precizeze dacă ele pot fi vârfurile unui triunghi. În caz afirmativ, să se spună de ce tip este triunghiul (oarecare, echilateral, isoscel, dreptunghic, dreptunghic isoscel) și să se calculeze aria și perimetrul triunghiului.
14. Se citesc coeficienții ecuațiilor carteziene generale a trei drepte. Să se analizeze relația dintre aceste drepte: se intersectează toate în același punct, se intersectează două câte două, sunt toate trei paralele, toate trei se suprapun, două sunt paralele între ele și se intersectează cu a treia, două se suprapun și se intersectează cu a treia sau două se suprapun și sunt paralele cu a treia. În cazul în care există puncte de intersecție, să li se afișeze coordonatele.
15. Se citesc un număr natural n și un număr real l care reprezintă lungimea laturii unui pătrat. Pătratului l se circumscrie un cerc, cercului un pătrat ș.a.m.d. Să se determine aria pătratului și a cercului obținute după n operații de circumscriere.

3.5.2. Rezolvarea problemelor de fizică

Enunțul problemei 1: Două mobile pornesc din punctul A, se mișcă rectiliniu uniform în aceeași direcție, cu vitezele v_1 și v_2 ($v_1 > v_2$), și ajung simultan în punctul B, la distanța d de A. Mobilul 1 pleacă mai târziu cu t_0 decât mobilul 2. Se consideră cunoscute: v_1 , v_2 și d . Să se calculeze t_0 . Unitatea de măsură pentru viteze este km/h, iar pentru distanță km. Timpul se va exprima în ore, minute și secunde.

Se notează cu d distanța dintre A și B, cu t_1 și t_2 timpul de mișcare pentru cele două mobile și cu v_1 și v_2 vitezele lor. Legea de mișcare pentru cele două mobile are forma:

$$(1) \quad d = v_1 \times t_1$$

$$(2) \quad d = v_2 \times t_2 = v_2 \times (t_1 + t_0)$$

Rezultă că $t_0 = d \times (1/v_1 + 1/v_2)$. Unitatea de măsură pentru distanță va fi m , pentru viteze m/s , iar pentru timp s .

Se vor folosi următoarele variabile de memorie;

✓ **Date de intrare:** v_1 , v_2 și d .

✓ **Date de ieșire:** th , tm și ts pentru reprezentarea timpului în ore, minute și secunde.

Algoritmul este:

```
real v1, v2, d, th, tm, ts;
inceput
citește v1, v2, d;
v1 ← v1*5/18; v2 ← v2*5/18; d ← d*1000;
th ← d*(1/v2-1/v1);
dacă th ≥ 3600
    atunci
        tm ← th - [(th/3600)]*3600; // [x] - parte întreagă din x
        th ← [(th/3600)];
    altfel
        tm ← th; th ← 0;
sfârșit_dacă;
dacă tm ≥ 60
    atunci ts ← tm - [(tm/60)]*60; tm ← [(tm/60)];
    altfel ts ← tm; tm ← 0;
sfârșit_dacă;
scrie "timp= ", th, "h ", tm, "m ", ts, "s";
sfârșit.
```

Enunțul problemei 2: Două trenuri de lungimi L_1 și L_2 și având vitezele v_1 și v_2 ($v_2 > v_1$) se apropie unul de altul în mișcare rectilinie uniformă, pe linii paralele. Se consideră cunoscute: v_1 , v_2 și $L_1 = L_2 = d$. Aflați timpul t scurs între momentul când se întâlnesc și momentul depășirii complete. Unitatea de măsură pentru viteze este km/h , iar pentru distanță m . Timpul se va exprima în secunde.

Se notează cu d lungimea trenului, cu t timpul și cu v_1 și v_2 vitezele trenurilor.

Unitatea de măsură pentru lungime va fi m , pentru viteze m/s , iar pentru timp s .

Există două variante:

Varianta 1. Trenurile se deplasează amândouă în aceeași direcție. Se notează cu x distanța parcursă de primul tren. Legea de mișcare pentru cele două trenuri are forma:

- (1) $x = v_1 t$
- (2) $x + 2d = v_2 t$

Rezultă că $t = 2d/(v_2 - v_1)$.

Varianta 2. Trenurile se deplasează în direcții opuse. Se notează cu x_1 distanța parcursă de primul tren și cu x_2 distanța parcursă de cel de al doilea tren. Legea de mișcare pentru cele două trenuri are forma:

- (1) $x_1 = v_1 t$
- (2) $x_2 = v_2 t$

unde $x_1 + x_2 = 2d$. Rezultă că $t = 2d/(v_1 + v_2)$.

Se vor folosi următoarele variabile de memorie;

- ✓ **Date de intrare:** v_1 , v_2 de tip real pentru viteze, d de tip întreg pentru distanță și opt de tip caracter, pentru a stabili care dintre variante se alege (are valoarea "1" pentru prima variantă și "2" pentru a doua variantă).
- ✓ **Data de ieșire:** t , de tip real, pentru calcularea timpului.

Algoritmul este:

```

real v1, v2, t;
întreg d;
caracter opt;
început
  citește v1, v2, d, opt;
  în caz că opt
    caz "1":  $t \leftarrow 2*d/((v_2-v_1)*(5/18))$ ;
             scrie t;
    caz "2":  $t \leftarrow 2*d/((v_2+v_1)*(5/18))$ ;
             scrie t;
    altfel scrie "Opțiune eronată";
  sfârșit în caz că;
sfârșit.

```

Probleme de fizică propuse:

16. Fie A și B două puncte pe dreaptă și d distanța dintre ele. Din A pleacă spre B un mobil cu viteza v_1 . După timpul t_0 pleacă din B, în același sens, un al doilea mobil cu viteza v_2 ($v_2 > v_1$). Se consideră cunoscute: v_1 , v_2 și t_0 . Se cere să se afle după cât timp t are loc întâlnirea vehiculelor și distanța x de la A la punctul de întâlnire. Unitatea de măsură pentru viteze este km/h, pentru distanță km, iar pentru timp h.
17. Fie A și B două puncte pe dreaptă și d distanța dintre ele. Din A și B pornesc în mișcare uniformă pe dreaptă, în același sens, două mobile cu vitezele v_1 și v_2 ($v_1 > v_2$), al doilea mobil cu timpul t_0 mai târziu decât primul. Se consideră cunoscute: v_1 , v_2 și t_0 . Se cere să se afle după cât timp t de la pornirea primului mobil, acesta îl ajunge pe al doilea și distanțele d_1 și d_2 parcurse de mobile până la întâlnirea lor. Unitatea de măsură pentru viteze este m/s, pentru distanță m, iar pentru timp s.
18. Fie A și B două puncte pe dreaptă și d distanța dintre ele. Din A pleacă spre B un mobil cu viteza v_1 și după timpul t_0 din B pleacă spre A un al doilea mobil cu viteza v_2 . Ambele mobile se deplasează în mișcare uniformă pe dreaptă și se întâlnesc la distanța x de B. Se consideră cunoscute: d , v_2 și t_0 . Se cere să se afle timpul t scurs de la plecarea primului mobil până la întâlnire și viteza v_1 . Unitatea de măsură pentru viteze este km/h, pentru distanță km, iar pentru timp s.
19. Două mobile pornesc simultan din punctul O în același sens, în mișcare uniformă pe dreaptă cu vitezele v_1 și v_2 ($v_1 > v_2$). După timpul t_1 , pornește din O, în același sens, un al treilea mobil. El ajunge primul mobil cu timpul t_2 mai târziu decât pe al doilea. Se consideră cunoscute: v_1 , v_2 , t_1 și t_2 . Se cere să se afle

viteza v_3 a mașinii a treia. Unitatea de măsură pentru viteze este km/h, iar pentru timp min.

20. Fie A și B două puncte pe o dreaptă și d distanța dintre ele. Din A și B pornesc simultan, unul spre altul, două mobile în mișcare uniformă. Ele se întâlnesc după timpul t_1 și își continuă fiecare mișcare. Mobilul plecat din A ajunge în punctul B cu timpul t_2 mai târziu decât ajunge în A mobilul plecat din B. Se consideră cunoscute: d , t_1 și t_2 . Se cere să se afle vitezele v_1 și v_2 ale mobilelor. Unitatea de măsură pentru viteze este m/s, pentru distanță m, iar pentru timp min.
21. Fie A și B două puncte pe o dreaptă și d distanța dintre ele. Din A și B pornesc simultan, unul spre altul, două mobile în mișcare uniformă. Ele se întâlnesc la distanța d_1 de B și își continuă fiecare mișcarea ajungând în B și respectiv în A. Se întorc fără oprire și se întâlnesc a doua oară la distanța d_2 de A la timpul t după prima întâlnire. Se consideră cunoscute: d_1 , d_2 și t . Aflați vitezele v_1 și v_2 ale mobilelor și distanța d . Unitatea de măsură este pentru viteze m/s, pentru distanță m, iar pentru timp s.
22. Fie A și B două puncte pe o dreaptă și d distanța dintre ele. Din A și B pornesc simultan, unul spre altul, două mobile în mișcare uniformă. Ele se întâlnesc la distanța d_1 de A și își continuă fiecare mișcarea ajungând în B și respectiv în A unde staționează timpul t , apoi se întorc cu aceeași viteză și se întâlnesc a doua oară la distanța d_2 de B. Se consideră cunoscute: d_1 , d_2 și t . Aflați distanța d dintre cele două puncte. Unitatea de măsură este pentru distanță km, iar pentru timp h.

Evaluare

Răspundeți:

1. Ce metode puteți folosi pentru reprezentarea unui algoritm?
2. De ce se folosesc cuvinte cheie la descrierea algoritmului prin pseudocod?
3. Dați câte un exemplu de problemă pentru fiecare dintre cele trei tipuri de structuri de control.
4. Verificați algoritmul lui Euclid de la pag. 58 pentru $a=2$ și $b=162$. Ce constatați? Modificați algoritmul astfel încât să fie mai eficient.

Adevărat sau Fals:

1. Pentru calculul modului unui număr x folosiți o structură repetitivă.
2. Pentru calculul produsului a 10 numere folosiți o structură alternativă.
3. Pentru a vedea dacă un element k aparține unei mulțimi infinite de elemente, se folosește un algoritm prin care se verifică fiecare element al mulțimii dacă este identic cu elementul k .
4. Pentru a vedea dacă un element j aparține unei mulțimi infinite de elemente, se construiește un algoritm prin care se verifică dacă elementul j corespunde unui criteriu prin care sunt definite elementele mulțimii.

Alegeți:

1. Fraza „cumpără acțiuni dacă produsul intern brut a crescut și vinde în caz contrar” poate fi descrisă cu o:
 - a) structură liniară
 - b) structură alternativă
 - c) structură repetitivă
2. Fraza „citește mai multe numere până când întâlnești numărul 0” poate fi descrisă cu:
 - a) o structură repetitivă cu număr cunoscut de pași
 - b) o structură alternativă generalizată
 - c) o structură repetitivă cu număr necunoscut de pași
3. La un magazin se acordă un bonus cumpărătorilor: dacă valoarea cumpărăturilor (v) este între 5.000.000 lei și 10.000.000 lei, bonusul este de 1% din valoare, dacă valoarea este între 10.000.000 lei și 20.000.000 lei, bonusul este de 2% din valoare, iar dacă valoarea este mai mare de 20.000.000 lei, bonusul este de 5% din valoare. Precizați care dintre pseudocoduri descrie corect algoritmul de acordare a bonusului¹:

d)

```

dacă a>5000000
atunci
  v←v-v/100;
sfârșit_dacă;
dacă v>10000000
atunci
  v←v-2*v/100;
sfârșit_dacă;
dacă v>20000000
atunci
  v←v-5*v/100;
sfârșit_dacă;

```

b)

```

dacă v>5000000
atunci
  dacă v>10000000
  atunci
    dacă v>20000000
    atunci
      v←v-5*v/100;
    altfel
      v←v-2*v/100;
    sfârșit_dacă;
  altfel
    v←v-v/100;
  sfârșit_dacă;
sfârșit_dacă;

```

c)

```

dacă v>5000000
atunci
  dacă v>10000000
  atunci
    dacă v>20000000
    atunci
      v←v-5*v/100;
    sfârșit_dacă;
  altfel
    v←v-2*v/100;
  sfârșit_dacă;
altfel
  v←v-v/100;
sfârșit_dacă;

```

4. În urma executării secvenței $p \leftarrow 2$; **pentru** $i=3,5$ **execută** $p \leftarrow p*i$; **sfârșit_pentru**; valoarea lui p este:
 - a) 16
 - b) 24
 - c) 120
5. În urma executării secvenței $s \leftarrow 10$; **pentru** $i=2,6$ **pas 2** **execută** $s \leftarrow s+i$; **sfârșit_pentru**; valoarea lui s este:
 - a) 16
 - b) 22
 - c) 30
6. În urma executării secvenței $p \leftarrow 2$; **pentru** $i=1,3$ **execută** $p \leftarrow p*p$; **sfârșit_pentru**; valoarea lui p este:
 - a) 16
 - b) 64
 - c) 256

¹ **Sugestie:** Verificați fiecare pseudocod alegând corect un set complet de date de intrare pentru testarea algoritmului. Alegerea setului complet de date de intrare se face dând valori corespunzătoare variabilei a pentru fiecare dintre cele patru cazuri de acordare a bonusului.

Algoritmi

de în caz

Festivá

poate fi

cumpără-

1% din va-

musul este de

ei, bonusul

correct algo-

4000

000000

$$= 2/100;$$

...că;

100;

...pentru;

$$S \leftarrow S + i;$$

parsit pentru;

de intrare

când valori

bonusului.

Informatică

7. În urma executării secvenței $i \leftarrow 3$; **cât timp** $i < 10$ **execută** $i \leftarrow i + 3$; **scrie** i ; **sfârșit_cât_timp**; se afișează numerele:
a) 3, 6, 9 b) 6, 9, 12 c) 6, 9
8. În urma executării secvenței $i \leftarrow 3$; **repetă** **scrie** i ; $i \leftarrow i + 3$ **până când** $i > 10$; se afișează numerele:
a) 3, 6, 9 b) 6, 9, 12 c) 6, 9
9. Pentru $n=3$ și $x=2$, în urma executării secvenței **pentru** $i=1, n$ **execută** $x \leftarrow x * i$; **sfârșit_pentru**; se afișează numerele:
a) 16 b) 256 c) 8
10. Pentru $n=3$ și $x=8$, în urma executării secvenței **pentru** $i=1, n$ **execută** $x \leftarrow x + i * i$; **sfârșit_pentru**; se afișează numerele:
a) 15 b) 38 c) 22
11. Fie $n=2033$. Ca în urma executării secvenței: **cât timp** $n \geq m$ **execută** $n \leftarrow n - m$; **sfârșit_cât_timp**; – variabila n să aibă valoarea 0, un număr m cu două cifre trebuie să aibă valoarea:
a) 99 b) 97 c) 19

Rezolvați:

- Rezolvați:**
12. Se consideră următorul algoritm reprezentat în pseudocod (n este un număr natural):

```

citește n; m ← 0;
cât timp n > 0 execută
    n ← n div 2; m ← m + 1;
sfârșit_cât_timp;
scrie m;

```

- Ce valoare se va afișa pentru $n=21$?
- Ce valori se pot citi pentru n astfel încât să se afișeze 2?
- Scriveți pseudocodul unei structuri repetitive condiționate posterior echivalente.

13. Se consideră următorul algoritm reprezentat în pseudocod (a și b sunt numere naturale):

```

citește a,b;
dacă a>b atunci x ← a; a ← b; b ← x;
sfârșit_dacă;
n ← 0;
pentru i = a,b execută
    dacă i mod 2 = 0 atunci n ← n+1
    sfârșit_dacă
sfârșit_pentru;
scrie n;

```

- Ce valoare se va afișa pentru $a=20$ și $b=29$?
- Precizați o valoare pentru a și o valoare pentru b astfel încât să se afișeze 0.
- Scriveți în pseudocod un algoritm echivalent care să folosească structura repetitivă condiționată anterior și un algoritm care să nu folosească nici o structură repetitivă.

14. Se consideră următorul algoritm reprezentat în pseudocod (a și b sunt numere naturale):

```

citește a,b;
dacă a>b atunci x ← a; a ← b; b ← x;
sfârșit_dacă;
n ← 0;
cât timp a<b execută
    a ← a+1; b ← b-1; n ← n+1;
sfârșit_pentru;
scrie n;

```

- Ce valoare se va afișa pentru a=17 și b=25?
 - Precizați o valoare pentru a și o valoare pentru b astfel încât să se afișeze 0.
 - Scrieți în pseudocod un algoritm echivalent care să nu folosească nici o structură repetitivă.
15. Să se calculeze funcția $\sin(x)$ folosind următoarea expresie iterativă:
- $$e_n = x - \frac{x^3}{(1 \cdot 2 \cdot 3)} + \frac{x^5}{(1 \cdot 2 \cdot 3 \cdot 4 \cdot 5)} - \frac{x^7}{(1 \cdot 2 \cdot 3 \cdot 4 \cdot 5 \cdot 6 \cdot 7)} + \dots + (-1)^n \frac{x^{2n+1}}{(1 \cdot 2 \cdot 3 \cdot 4 \cdot 5 \cdot 6 \cdot 7 \dots (2n+1))}$$
- Calculul iterativ al expresiei $e(x)$ se va face până când $|e_n - e_{n-1}| < 0.00001$. Valoarea lui e_n va fi cu o aproximație de 0.00001 valoarea lui $\sin(x)$.
16. Să se calculeze funcția $\cos(x)$ folosind următoarea expresie iterativă:
- $$e_n = 1 - \frac{x^2}{(1 \cdot 2)} + \frac{x^4}{(1 \cdot 2 \cdot 3 \cdot 4)} - \frac{x^6}{(1 \cdot 2 \cdot 3 \cdot 4 \cdot 5 \cdot 6)} + \dots + (-1)^n \frac{x^{2n}}{(1 \cdot 2 \cdot 3 \cdot 4 \cdot 5 \cdot 6 \cdot 7 \dots (2n))}$$
- Calculul iterativ al expresiei $e(x)$ se va face până când $|e_n - e_{n-1}| < 0.00001$. Valoarea lui e_n va fi cu o aproximație de 0.00001 valoarea lui $\cos(x)$.
17. Următorii doi algoritmi descriși în pseudocod calculează a^b , unde a și b sunt două numere naturale. Analizați cei doi algoritmi din punct de vedere al eficienței. Precizați care este algoritmul cel mai eficient. Justificați răspunsul.

Varianta 1

```

întreg a,b,i,p;
început
    citește a,b;
    i ← b; p ← 1;
    cât timp i>0 execută
        p ← p*a;
        i ← i-1;
    sfârșit_cât_timp;
    scrie p;
sfârșit.

```

Varianta 2

```

întreg a,b,i,p;
început
    citește a,b;
    i ← b; p ← 1;
    cât timp i>0 execută
        dacă i mod 2 = 0
            atunci
                p ← p*a*a; i ← i-2;
            altfel
                p ← p*a; i ← i-1;
        sfârșit_dacă;
    sfârșit_cât_timp;
    scrie p;
sfârșit.

```

4. Implementarea algoritmilor

4.1. Caracteristicile limbajului de programare

Pentru a putea executa cu ajutorul calculatorului algoritmi descriși în pseudocod, **aceștia trebuie implementați într-un limbaj de programare**, adică trebuie să-i re-prezentați cu ajutorul instrucțiunilor unui limbaj de programare, de exemplu, **limbajul C++**. Transcrierea algoritmului într-un limbaj de programare se face cu un **program specializat numit editor de texte**. În acest mod se obține un fișier care **conține un text și care se numește program sursă**. În limbajul C++ fișierul care **conține programul sursă are extensia .cpp**.

Pentru a putea rezolva o problemă cu ajutorul calculatorului trebuie însă să folosiți **un program executabil** (un program care să fie încărcat în memoria internă a calculatorului și care să poată fi executat de procesor). Aceasta înseamnă că **nu este suficient să transcrieți algoritmul din limbajul pseudocod în limbajul de programare, ci trebuie să mai executați și alte operații**:

- ✓ **Compilarea**, adică traducerea fiecărei instrucțiuni din limbajul de programare într-un grup de instrucțiuni în limbajul mașinii, obținându-se **programul obiect**. Programul obiect se memorează tot într-un fișier, care are însă extensia **.obj**.
- ✓ **Editarea legăturilor**, adică asamblarea pieselor rezultate în urma compilării pentru a se obține **programul executabil**. În programul sursă pe care îl scrieți veți face apel de multe ori la funcțiile din bibliotecile standard ale limbajului. Aceste funcții au fost realizate și implementate în limbajul de programare de către autorii lui și se numesc **funcții standard**. Ele realizează operații de care aveți nevoie într-un algoritm, cum sunt de exemplu operațiile de citire sau scriere (funcțiile de citire și de scriere) sau operații matematice (funcțiile matematice). Programul executabil trebuie să conțină, pe lângă grupurile de instrucțiuni în cod mașină obținute în urma traducerii instrucțiunilor din programul sursă, și instrucțiunile în cod mașină ale funcțiilor standard pe care le-ați apelat în program. Rolul operației de editare a legăturilor este acela de a asambla împreună instrucțiunile în cod mașină obținute prin traducerea instrucțiunilor din programul sursă cu instrucțiunile în cod mașină ale funcțiilor standard apelate în program.

Aceste operații se realizează cu ajutorul a două programe numite **compilator și editor de legături**. Ansamblul de programe compilator și editor de legături funcționează ca un translator între două persoane care vorbesc limbi diferite. Dacă în cazul unui translator traducerea se face între două limbi naturale, în cazul acestor programe **traducerea se face între două limbi artificiale: limbajul de programare și limbajul mașinii**. Compilatorul reprezintă dicționarul pe baza căruia se realizează traducerea cuvintelor, iar editorul de legături assemblează aceste cuvinte în construcții gramaticale corecte.

La fel ca și un limbaj natural, orice limbaj de programare este caracterizat de:

- ✓ **Sintaxă.** Este formată din totalitatea regulilor de scriere corectă astfel încât să se realizeze construcții acceptate de compilator (construcții pe care compilatorul să le poată traduce în cod mașină). Dacă nu veți realiza construcții corecte care să-i permită compilatorului să identifice și să recunoască fiecare unitate lexicală, nu veți putea transmite calculatorului informațiile necesare pentru rezolvarea problemei (algoritmul și datele cu care lucrează algoritmul).
- ✓ **Semantică.** Reprezintă semnificația construcțiilor corecte din punct de vedere sintactic. Chiar dacă ați realizat o construcție corectă din punct de vedere lexical, trebuie să cunoașteți semnificația ei pentru a fi siguri că ați implementat corect algoritmul pentru rezolvarea problemei.
- ✓ **Vocabular.** Este format din totalitatea cuvintelor care pot fi folosite într-un program.

Setul de caractere folosit pentru construirea cuvintelor este format din:

- ✓ **literele mari și mici ale alfabetului latin** ($a \div z$, $A \div Z$),
- ✓ **cifrele sistemului de numerație zecimal** ($0 \div 9$),
- ✓ **caracterele speciale** (spațiu, +, -, *, /, %, =, !, #, ", (,), [,], ;, :, ., etc.)

Cuvintele pot fi: **identificatori**, **separatori**, **constante**, **cuvinte cheie** și **operatori**.

Identificatorii sunt nume de obiecte folosite în program (cum sunt, de exemplu, variabilele de memorie și funcțiile). Identificatorul este o succesiune de litere, cifre sau caracterul de subliniere (`_`), din care prima trebuie obligatoriu să nu fie cifră. Lungimea maximă a unui identificator este de 31 de caractere. De exemplu, sunt considerați identificatori corecți: *a1*, *A1*, *delta*, *_nr*, *nr_prim*, dar nu sunt considerați corecți identificatorii *1A*, *A B*, *A&B* sau *c.m.m.d.c.*. Limbajul C++ face **diferență între literele mari și literele mici** ale alfabetului. De exemplu, identificatorul *A1* este diferit de identificatorul *a1*.

Recomandări:

1. Nu începeți un nume de variabilă de memorie cu caracterul `_`, deoarece în funcțiile standard se folosesc frecvent identificatori care încep cu acest caracter.
2. Folosiți literele mici pentru numele de variabile și literele mari pentru numele de constante.

Separatorii se folosesc pentru a delimita unitățile lexice (o unitate lexicală este formată dintr-un șir de caractere care are o semnificație lingvistică). Se folosesc în rolul de separatori:

- ✓ spațiul și caracterul tabulator (TAB) pentru a separa cuvintele,
- ✓ `;` pentru a separa instrucțiunile (corespunde punctului care separă propozițiile din limbajul natural),
- ✓ ansamblul de caractere de control CR+LF generat la apăsarea tastei **Enter** pentru a separa liniile de text (corespund scrierii unui paragraf de la începutul rândului din limbajul natural),
- ✓ virgula (`,`) care separă elementele unei liste (corespunde virgulei care separă cuvintele unei enumerări din limbajul natural).

Cuvintele cheie (*keywords*) sunt identificatori cu o semnificație precisă pentru limbaj, care nu pot fi folosiți într-un alt context decât cel permis de semantica limbajului.

De exemplu, `int`, `char`, `float`, `double`, `unsigned`, `signed`, `short`, `long`, `void`, `sizeof`, `typedef`, `struct`, `const`, `enum`, `if`, `else`, `switch`, `case`, `default`, `for`, `while`, `do`, `break`, `continue`, `return` etc. Din această cauză nu puteți folosi ca identificatori cuvinte cheie.

Programul poate conține și comentarii. **Comentariile** sunt texte explicative folosite în program pentru a-l face mai ușor de înțeles. Ele nu sunt interpretate de compilator. Pot fi scrise oriunde în program și sunt delimitate de restul programului astfel:

- ✓ dacă ocupă mai multe rânduri, se folosesc perechile de caractere `/*` și `*/`,
- ✓ dacă se scrie pe un singur rând, comentariul este precedat de caracterele `//`.

De exemplu:

```
a)  /* Acesta este  
      un comentariu */  
b)  // Acesta este  
      // un comentariu
```

4.2. Structura programului

Pentru a înțelege structura unui program, vom defini mai întâi **blocul**.

Blocul este unitatea de bază a oricărui program C++.

Blocul este format din linii de text scrise în limbajul de programare. Pe o linie de text se pot scrie una sau mai multe **instrucțiuni**. Instrucțiunea codifică în limbajul de programare un pas al algoritmului descris în pseudocod. Blocul poate conține două părți:

- ✓ **Partea declarativă.** Conține definiții de obiecte necesare algoritmului pentru rezolvarea problemei: tipuri de date, constante, variabile de memorie etc. Definirea lor se face cu ajutorul **instrucțiunilor declarative**. Instrucțiunile declarative furnizează compilatorului informații despre semnificația identificatorilor folosiți în program.

- ✓ **Partea executabilă sau partea procedurală.** Conține instrucțiunile care descriu pașii algoritmului care trebuie implementat pentru rezolvarea problemei. Aceste instrucțiuni se numesc **instrucțiuni imperative**. Ele sunt:

- a) **Instrucțiunea expresie** prin care se evaluează o expresie. În cazul în care expresia conține operatorul de atribuire prin care se atribuie unei variabile de memorie valoarea unei expresii, ea va fi echivalentă cu o **instrucțiune de atribuire** din alte limbaje de programare și corespunde operației de atribuire din algoritm. În cazul în care expresia conține numai apelul unei funcții (funcția este un program care poate fi apelat dintr-un alt program, apelarea ei însemnând o cerere de executare), ea va fi echivalentă cu o **instrucțiune procedurală** din alte limbaje de programare și corespunde unei operații complexe din algoritm (cum este, de exemplu, scrierea sau citirea datelor).
- b) **Instrucțiunea de control** prin care se modifică ordinea de execuție a programului. Ea corespunde unei structuri de control dintr-un algoritm.

Observație. La sfârșitul fiecărei instrucțiuni se scrie caracterul separator;

Limbajul de programare permite definirea unei instrucțiuni compuse. **Instrucțiunea compusă** este o instrucțiune formată dintr-un grup de instrucțiuni care sunt interpretate de compilator ca o singură instrucțiune. Definirea se face prin încadrarea grupului de instrucțiuni de parantezele { ... }. Parantezele acoladă sunt **separatori**.

Blocul este încapsulat într-o instrucțiune compusă. Nu este obligatoriu ca blocul să conțină ambele părți. Construcții de forma { } sau { ; } sunt acceptate de compilator și înseamnă **blocul vid** (blocul care nu conține nimic), respectiv blocul care conține o instrucțiune care nu face nimic, adică **instrucțiunea vidă**.

Orice program C++ este o **colecție de definiții de variabile și funcții**. **Funcția** are un nume și rezolvă o anumită sarcină. Ea este un bloc precedat de un **antet**. Așadar programul conține entități de forma:

```
antet funcție
{ bloc }
```

În **antet** se precizează numele funcției, tipul rezultatului pe care-l întoarce, prin chiar numele său, și eventual parametrii de execuție (valori care se transmit blocului și care sunt necesare când se execută):

tip_rezultat nume (listă_parametri)

Dacă nu se precizează **tip_rezultat** se presupune că funcția întoarce un rezultat întreg.

Una dintre funcții este **funcția rădăcină**. Ea este obligatorie și este primul bloc cu care începe execuția programului. Numele său este **main()**. Antetul acestei funcții este:

```
void main()
```

ceea ce semnifică faptul că funcția nu întoarce nici un rezultat (**void**) și nu necesită parametri pentru apelare – parantezele () nu conțin o listă de parametri. Chiar dacă funcția nu are nevoie de parametri pentru apelare, parantezele () sunt obligatorii. Dacă nu se precizează pentru funcția **main** tipul rezultatului, chiar dacă funcția nu întoarce nici un rezultat, nu se va produce o eroare de compilare, ci numai un avertisment al compilatorului. Pentru algoritmii simpli pe care îi veți implementa cu ajutorul limbajului C++ nu veți folosi decât o singură funcție: funcția rădăcină.

Ca exemplu pentru structura unui program C++ se consideră următoarea problemă:

Enunțul problemei: Se citesc două numere întregi **a**, **b**. Să se calculeze suma lor.

Programul în limbajul C++ este:

void main()	Antetul funcției rădăcină (este obligatoriu)
{	Începutul blocului (este obligatoriu)
int a,b,s;	Partea declarativă; conține o instrucțiune declarativă pentru variabilele de memorie a , b și s , de tip întreg.
	Începutul părții executabile
cout<<"a= "; → cin>>a; cout<<"b= "; cin>>b; s=a+b; → cout<<"suma= "<<s;	Operațiile de citire și scriere se execută cu instrucțiuni expresie care creează fluxuri de date implementate în bibliotecile limbajului de programare (cin>> și cout<<). Corespund unei instrucțiuni procedurale din alte limbaje. Instrucțiune expresie pentru operația de atribuire.
}	Sfârșitul blocului (este obligatoriu)

4.3. Instrucțiunile declarative

Se folosesc pentru declararea obiectelor folosite în algoritm:

- ✓ variabile de memorie,
- ✓ constante simbolice,
- ✓ tipuri de date.

4.3.1. Tipuri de date

Ați aflat deja că unul dintre atributele unei date elementare este **tipul ei**. În limbajul C++ puteți folosi următoarele tipuri de date:

- ✓ **tipuri standard** sau **predefinite**, adică tipuri implementate în limbaj, și
- ✓ **tipuri definite** de utilizator.

Fiecare tip de dată se identifică printr-un nume. Identificatorii folosiți pentru tipurile **standard** sunt cuvinte cheie. În limbajul C++ sunt implementate următoarele **tipuri standard (tipuri de bază)**:

- ✓ **char** pentru memorarea caracterelor;
- ✓ **int** pentru memorarea numerelor întregi;
- ✓ **float** și **double** pentru memorarea numerelor reale;
- ✓ **void** pentru tip neprecizat.

Observație: În limbajul C++ **nu este implementat tipul logic**. Pentru acest tip de dată se poate folosi orice tip numeric. Interpretarea este următoarea:

- ✓ unei expresii al cărei rezultat este de tip logic i se atribuie valoarea **0** pentru *False* și **1** pentru *True*;
- ✓ într-o expresie, un operand numeric este interpretat ca un operand logic, astfel: dacă are valoarea **0**, este considerat *False*, iar dacă are valoarea **diferită de 0**, este considerat *True*.

Tipul datei determină dimensiunea zonei de memorie alocată datei și modul de codificare și, implicit, **domeniul de definiție intern al datei**. Dimensiunea zonei de memorie alocată unei date de un anumit tip este dependentă de mașină (de hardware), cu excepția tipului **char** pentru care se folosește 1 octet. De exemplu, pentru tipul **int** dimensiunea poate fi de 2 octeți (16 biți) sau de 4 octeți (32 de biți). Compilatorul C++ alege singur dimensiunea specifică echipamentului hardware.

Modurile de codificare folosite pentru tipurile de bază sunt:

- ✓ **char** folosește reprezentarea caracterului în codul ASCII (care este un cod numeric),
- ✓ **int** folosește reprezentarea în complement față de 2,
- ✓ **float** folosește reprezentarea în virgulă mobilă simplă precizie,
- ✓ **double** folosește reprezentarea în virgulă mobilă dublă precizie.

Aceste moduri de codificare sunt prezentate în Anexa 3.

Tipul char este tratat astfel:

- ✓ **operația de atribuire:** i se poate atribui o constantă de tip caracter sau o valoare numerică ce reprezintă **codul ASCII** al unui caracter.

- ✓ **operatori matematici:** într-o expresie se pot aplica operatori matematici pe acest tip de dată, în evaluarea expresiei fiind folosită valoarea numerică memorată în dată.
- ✓ **operațiile de citire și scriere:** valoarea introdusă de la tastatură nu poate să fie decât un caracter, iar valoarea numerică memorată în dată este afișată sub forma unui caracter al cărui cod ASCII este acea valoare numerică.

Modul de implementare a acestor tipuri de date poate fi modificat prin declararea suplimentare de tip: **short** (scurt), **long** (lung), **signed** (cu semn) și **unsigned** (fără semn). De exemplu, tipul **unsigned char** este tipul **char** pe care s-a aplicat modificatorul **unsigned** și înseamnă tipul caracter fără semn, adică valoarea numerică memorată care reprezintă codul ASCII al unui caracter este interpretată ca un număr întreg pozitiv, sau 0, iar tipul **signed char** este tipul **char** pe care s-a aplicat modificatorul **signed** și înseamnă tipul caracter cu semn, adică valoarea numerică memorată este interpretată ca un număr întreg pozitiv sau negativ.

Observații:

1. Toți acești **modificatori de tip** se pot aplica pe tipul **int**. În acest caz, tipul **int** poate fi omis (este considerat implicit). Astfel, tipul **short** este echivalent cu tipul **short int**, iar tipul **unsigned long** este echivalent cu tipul **unsigned long int**. Deoarece tipul **int** se folosește pentru numere întregi (pozitive și negative), aplicarea modificatorului **signed** pe acest tip de dată este de prisos.
2. Pe tipul **char** se pot aplica numai modificatorii **signed** și **unsigned**.
3. Pe tipurile de date reale se poate aplica numai modificatorul **long**, și acesta numai pe tipul **double**.
4. Prin aplicarea modificatorilor de tip se modifică dimensiunea zonei de memorie alocată acelui tip. Regulile sunt următoarele: pentru tipul **short** cel puțin 2 octeți, dar nu mai mult decât pentru tipul **int**, iar pentru tipul **long**, cel puțin 4 octeți, dar nu mai puțin decât pentru tipul **int**. Astfel, dacă pe mașina respectivă pentru tipul **int** dimensiunea zonei de memorie alocată este de 2 octeți, aplicarea modificatorului **short** pe acest tip de dată este de prisos.

În tabelul următor sunt prezentate tipurile de date pe care le puteți folosi în cazul unei mașini pentru care tipul **int** este reprezentat pe 2 octeți:

Tip de dată	Dimensiune	Domeniu de definiție intern al datei
char	8 biți	-128÷127
unsigned char	8 biți	0÷255
int	16 biți	-32.768÷32.767
unsigned	16 biți	0÷65.535
long	32 biți	-2.147.483.648÷2.147.483.647
unsigned long	32 biți	0÷4.294.967.205
float	32 biți	$\pm 3,4 \cdot 10^{-37}$ ÷ $\pm 3,4 \cdot 10^{37}$ (cu o precizie de 6 cifre)
double	64 biți	$\pm 1,7 \cdot 10^{-308}$ ÷ $\pm 1,7 \cdot 10^{308}$ (cu o precizie de 10 cifre)
long double	80 biți	$\pm 3,4 \cdot 10^{-4932}$ ÷ $\pm 1,1 \cdot 10^{4932}$ (cu o precizie de 15 cifre)

Cunoașterea domeniului intern de definiție al tipului de dată este importantă în alegerea corectă a tipului de dată. De exemplu, dacă cerința este ca într-o dată să se memoreze o valoare numerică din domeniul $[10.000; 40.000]$, se va folosi tipul **unsigned int**, iar dacă cerința este ca într-o dată să se memoreze un număr natural cu 9 cifre, se va folosi tipul **unsigned long int**.

4.3.2. Constantele

Într-un program puteți folosi:

- ✓ **constante literale** – sunt valori efective pe care le folosiți în program;
- ✓ **constante simbolice** – sunt identificatori pe care îi folosiți în locul valorilor efective.

Constantele literale pot fi:

- ✓ numerice: – întregi,
– reale,
- ✓ caracter,
- ✓ șir de caractere.

Constantele numerice literale

Observație. Constantele numerice întregi nu pot fi decât numere pozitive.

Constantele numerice întregi pot fi exprimate:

- ✓ în baza 10, printr-un număr întreg pozitiv (de exemplu: 2, 1021),
- ✓ în baza 8, printr-un număr întreg pozitiv precedat de un 0 (de exemplu: 0124 reprezintă $124_{(8)}$),
- ✓ în baza 16, printr-un număr întreg pozitiv precedat de 0x sau 0X (de exemplu: 0X1A sau 0x1a reprezintă $1A_{(16)}$).

Constantele numerice reale pot fi exprimate:

- ✓ printr-un număr real, separarea părții întregi de partea fracționară făcându-se cu caracterul punct (de exemplu -50.12; 2.; .01; 0.01),
- ✓ folosind **notația științifică**, în care numerele sunt reprezentate prin **mantisă** și **exponent**, cu ajutorul puterilor lui 10 (exponenții), cele două părți fiind separate de litera **e** sau **E** (de exemplu -5 E -10, care înseamnă $-5 \cdot 10^{-10}$, sau -1.1e -5 care înseamnă $-1.1 \cdot 10^{-5}$).

$$12.5 = 125 \cdot 10^{-1} = 125 \text{ E } -1 = 125 \text{ e } -1$$

mantisă

exponent

Constantele caracter literale

Pot fi exprimate:

- ✓ printr-un caracter delimitat cu apostrofuri (de exemplu: 'a', 'A', '1'),
- ✓ prin codul ASCII al caracterului, precizat în baza 10 (de exemplu: 97, 65, 49),
- ✓ prin **secvențe escape**, constanta fiind precizată prin codul său ASCII exprimat în baza 8 sau 16; secvența escape este delimitată de apostrofuri și începe cu carac-

terul \ (de exemplu, caracterul 'a' are codul ASCII $97_{(10)} = 141_{(8)} = 61_{(16)}$ și constanta 'a' mai poate fi reprezentată și prin secvențele escape '\141' sau '\x61').

Observație. Secvențele escape se folosesc în general pentru:

- ✓ caractere care sunt reprezentate în codul ASCII dar nu pot fi scrise prin delimitarea cu apostrofuri a unor caractere introduse de la tastatură, așa zisele **caractere albe** (*whitespace*), pentru aceste caractere putând fi folosite și **notațiile speciale**:

Caracterul	Reprezentat în:			
	notație specială	cod ASCII în:		
		zecimal	octal	hexazecimal
newline (linie nouă)	'\n'	10	'\12'	'\xA'
carriage return (retur de car)	'\r'	13	'\15'	'\xD'
bell (semnal sonor)	'\a'	7	'\7'	'\x7'
tabulator orizontal	'\t'	9	'\11'	'\x9'
tabulator vertical	'\v'	11	'\13'	'\xB'
backspace (revenire cu o poziție)	'\b'	8	'\10'	'\x8'
formfeed (salt de pagină la imprimantă)	'\f'	12	'\14'	'\xC'

- ✓ caractere care au o semnificație specială în definirea constantelor de tip caracter, pentru aceste caractere putând fi folosite și **notațiile speciale**:

Caracterul	Reprezentat în:			
	notație specială	cod ASCII în:		
		zecimal	octal	hexazecimal
\ (backslash)	'\\'	92	'\134'	'\x5C'
' (apostrof)	'\''	34	'\47'	'\x27'
? (semn de întrebare)	'\?'	63	'\65'	'\x35'

Constantele șir de caractere literale

O constantă de tip șir de caractere este formată dintr-un grup de caractere delimitat cu ghilimele (de exemplu "abcd", "ABC", "123"). Atunci când vrei să afișezi pe ecran un mesaj, veți scrie o constantă de tip șir de caractere.



Un caracter delimitat de apostrofuri este diferit de un caracter delimitat de ghilimele (de exemplu 'a' este diferit de "a"). Primul este o constantă de tip **char** (o dată elementară), iar al doilea este o structură de date denumită șir de caractere ce conține un singur caracter.

4.3.3. Declararea variabilelor de memorie

Declararea variabilelor de memorie se face prin instrucțiunea:

```
tip_dată nume;
```

unde **tip_dată** precizează tipul datei memorate în variabila de memorie, iar **nume** identificatorul variabilei de memorie. De exemplu, prin instrucțiunile declarative:

```
int a;
```

```
char b;
```

se declară două variabile de memorie: *a* de tip întreg și *b* de tip caracter.

La declarare, variabilei de memorie *i* se alocă o zonă de memorie cu dimensiunea corespunzătoare tipului de dată, iar tipul de dată va determina modul în care va putea fi prelucrată acea variabilă de memorie în program.

Observații:

1. Într-o instrucțiune declarativă se pot declara mai multe variabile de memorie de același tip, astfel:

```
tip_dată listă_nume;
```

unde **listă_nume** este o listă de nume de variabile de memorie de același tip, în care numele sunt separate prin virgulă. De exemplu, prin instrucțiunile declarative:

```
int a,b,c;
```

```
char x,y;
```

se declară variabilele de memorie: *a*, *b* și *c* de tip întreg și *x* și *y* de tip caracter.

2. La declararea unei variabile de memorie aceasta va avea o **valoare reziduală** (reprezintă valoarea atribuită zonei de memorie alocate variabilei de către un program care a fost executat anterior). Prin instrucțiunea declarativă, variabilei de memorie *i* se poate atribui o valoare inițială (poate fi **inițializată**), astfel:

```
tip_dată nume=valoare;
```

unde **valoare** reprezintă o constantă de același tip ca și variabila de memorie sau o variabilă de memorie declarată anterior și căreia *i* s-a atribuit o valoare, iar caracterul egal (=) are rol de **separator** între numele variabilei de memorie și valoarea cu care se inițializează. De exemplu, prin instrucțiunile declarative:

```
int a, b=1, c=2;
```

```
float d=10;
```

```
char x='a', y=97, z='141', w='x61';
```

se declară variabilele de memorie de tip întreg: *a* care are o valoare reziduală, *b* care are valoarea 1 și *c* care are valoarea 2, o variabilă de tip real *d* care are valoarea 10 și patru variabile de memorie de tip caracter *x*, *y*, *z* și *w* care conțin o valoare numerică întreagă (97) care reprezintă codul ASCII al literei *a*.



O instrucțiune declarativă poate să apară oriunde în blocul funcției (nu este obligatoriu să fie scrisă la începutul blocului). **Regula** care trebuie însă respectată este: **declararea unei date trebuie să se facă înainte ca ea să fie folosită într-o instrucțiune imperativă.**

Prin instrucțiunile declarative:

```
char x='a';
```

```
int a=10, b=a, c=x;
```

se declară variabila de memorie de tip caracter *x* care conține o valoare numerică întreagă (97) care reprezintă codul ASCII al literei *a* și trei variabile de memorie de tip întreg: *a* care are valoarea 10, *b* care are valoarea ce a fost atribuită anterior variabilei *a*, adică 10, și *c* care are valoarea ce a fost atribuită anterior variabilei *x*, adică 97.

algoritmilor

= 61₍₁₆₎ și
sau 'x61').

prin delimi-
zisele **carac-**
și **notațiile**

in:
hexazecimal
'xA'
'xD'
'x7'
'x9'
'xB'
'x8'
'xC'

de tip carac-

in:
hexazecimal
'x5C'
'x27'
'x35'

caractere delimitat
afășați pe ecran

ferit de un ca-
este diferit de
o dată elemen-
denumită șir de

memorie, iar **nume**,
declarative:



Ordinea în care se execută operațiile de declarare și inițializare a variabilelor din lista unei instrucțiuni declarative nu face parte din definiția limbajului și este stabilită de compilator. Variabila *c* se poate inițializa cu valoarea lui *x*, deoarece instrucțiunile se execută secvențial. Inițializarea variabilei *b* cu valoarea lui *a* depinde de ordinea în care compilatorul creează și inițializează variabilele dintr-o listă.

4.3.4. Declararea constantelor simbolice

Declararea constantelor simbolice se face prin instrucțiunea:

```
const [tip_dată]1 nume=valoare;
```

unde **const** este un cuvânt cheie care înseamnă definirea unei constante simbolice, **tip_dată** precizează tipul datei constante, **nume** este identificatorul constantei, iar **valoare** este valoarea constantei. Tipul de dată poate fi omis. În acest caz, constanta este considerată de tip **int**. De exemplu, prin instrucțiunile declarative:

```
const int A=5;      sau      const A=5;
```

```
const float PI=3.14;
```

```
float pi=3.14;
```

se declară două constante: *A* de tip întreg care are valoarea 5 și *PI* de tip real care are valoarea 3,14, și o variabilă de memorie de tip real *pi*.

Observație: Valoarea unei constante simbolice nu poate fi modificată printr-o operație de citire de la tastatură sau prin atribuirea unei expresii, dar poate fi afișată printr-o operație de scriere pe ecran sau poate fi folosită într-o expresie ca operand. Variabila de memorie *pi* a fost inițializată cu valoarea 3,14. Valoarea ei va putea fi modificată fie printr-o operație de citire de la tastatură, fie printr-o operație de atribuire, spre deosebire de constanta *PI* a cărei valoare nu poate fi modificată.

În cazul în care o valoare constantă apare de mai multe ori într-un program, se recomandă folosirea constantei simbolice în locul constantei literale, din următoarele motive:

- ✓ Programul este mai ușor de citit.
- ✓ La actualizarea programului, modificarea valorii constante se face mai ușor. În loc să se caute toate aparițiile constantei literale în program pentru a-i modifica valoarea, se modifică valoarea constantei simbolice în instrucțiunea declarativă. De exemplu, dacă pentru numărul π vrei să folosești valoarea 3,141593 în locul valorii 3,14, modificarea programului se face foarte simplu, prin modificarea valorii constantei simbolice *PI*.

Declararea unui set de constante

Un grup de constante întregi din domeniul [0, 32767] poate fi definit printr-o singură instrucțiune declarativă, sub forma unui **set de constante**, astfel:

¹ **Convenție de notare:** un element încadrat de paranteze [] poate fi omis, iar simbolul | pus între două elemente are semnificația conjuncției **sau**: primul element sau al doilea element.

```
enum [nume_set] {listă_constanta};
```

Setul de constante se mai numește și **constantă de tip enumerare**. Lista conține elemente de forma **nume=[valoare]** separate prin virgulă. Dacă se precizează valoarea, constanta va avea valoarea precizată. Dacă nu se precizează valoarea, constanta va avea valoarea constantei precedente în listă, incrementată cu 1, valoarea primei constante din listă fiind 0.

De exemplu:

enum logic {NU,DA} – s-a definit setul de constante *logic* care conține constantele *NU*, care are valoarea 0, și *DA*, care are valoarea 1.

enum set1 {A,B,C} – s-a definit setul de constante *set1* care conține constantele *A=0*, *B=1* și *C=2*.

enum set2 {A=2,B=5,C} – s-a definit setul de constante *set2* care conține constantele *A=2*, *B=5* și *C=6*.

enum set3 {A,B=2,C=4} – s-a definit setul de constante *set3* care conține constantele *A=0*, *B=2* și *C=4*.

enum set4 {A=5,B,C=B,D} – s-a definit setul de constante *set4* care conține constantele *A=5*, *B=6*, *C=6* și *D=7*.

enum caractere {BELL='\a',TAB='\t',BS='\b',RAND_NOU='\n',RAND_NOU='\r'} – s-a definit setul de constante *caractere* care conține constante pentru câteva dintre caracterele albe.

4.3.5. Declararea tipurilor de dată utilizator

Pe lângă tipurile predefinite de date, se pot defini și tipuri utilizator, prin instrucțiunea declarativă:

```
typedef definiție_tip nume_tip;
```

Observație: Tipul de dată trebuie definit înainte de a se declara o dată de acel tip (variabilă de memorie sau constantă). De exemplu:

```
typedef unsigned char byte;
```

```
.....
```

```
byte a;
```

S-a definit tipul *byte*, pentru definirea sa folosindu-se tipul de bază **unsigned char** și s-a declarat apoi variabila *a* cu tipul *byte*.

```
typedef enum {NU, DA} boolean;
```

```
.....
```

```
boolean x;
```

S-a definit tipul *boolean*, pentru definirea sa folosindu-se un set de constante, și s-a declarat apoi variabila *x* cu tipul *boolean*. Variabila de memorie *x* nu poate lua decât valorile *DA* și *NU*.

Observație: Declararea unui tip de dată nu creează un tip nou de dată, ci numai un nume nou care este atribuit unui tip existent și care poate fi folosit la declararea variabilelor de memorie și a constantelor simbolice. Tipul de dată la care se referă un tip utilizator este sinonim cu cel predefinit care a fost folosit la declararea lui, iar variabilele de memorie și constantele simbolice care au acest tip au aceleași pro-

prietăți ca și cele care au tipul predefinit din declarație. Folosirea tipurilor de date utilizator face mai ușor de citit un program atunci când trebuie modificat.

4.4. Operațiile de citire și scriere

În program se prelucrează datele conform algoritmului definit pentru rezolvarea problemei. În urma acestui proces de prelucrare, din datele de intrare se obțin datele de ieșire. Introducerea datelor și afișarea rezultatelor sunt sarcini importante ale unui program. În C++ toate operațiile de intrare-ieșire se execută prin fluxuri de date. Un **flux de date** (*stream*) reprezintă fluxul datelor de la sursă la destinație.

Sursa poate fi:

- ✓ tastatura,
- ✓ fișier pe disc,
- ✓ variabilă de memorie.

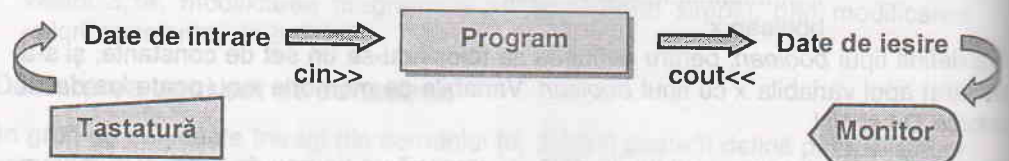
Destinația poate fi:

- ✓ ecranul monitorului,
- ✓ fișier pe disc,
- ✓ variabilă de memorie.

Fluxul de date este format din secvențe de caractere organizate pe **linii** (separate prin codurile ASCII generate la apăsarea tastei **Enter**) care intră și care ies din program. Prin fluxurile de date echipamentele de intrare-ieșire sunt conectate la programul C++.

Puteți folosi două **fluxuri de date standard**:

1. **Flux de date de intrare (cin)** care conectează tastatura la program, prin intermediul căruia se pot executa operațiile de **citire** prin care datele de intrare sunt furnizate programului. În acest caz fluxul datelor este între:
 - ✓ **sursă** – tastatura, și
 - ✓ **destinație** – variabila de memorie.
2. **Flux de date de ieșire (cout)** care conectează monitorul la program, prin intermediul căruia se pot executa operații de **scriere** prin care datele de ieșire sunt furnizate de program. În acest caz fluxul datelor este între:
 - ✓ **sursă** – variabila de memorie, și
 - ✓ **destinație** – ecranul monitorului.



Pentru operațiile de citire și scriere se folosesc instrucțiunile expresie prin care se creează fluxurile de date, cu ajutorul operatorilor >>, respectiv <<:

- ✓ **cin>>** – pentru citire, și
- ✓ **cout<<** – pentru scriere.

purilor de date
ficat.

rezolvarea pro-
se obțin datele
importante ale
prin fluxuri de
destinație.

linii (separate
și care ies din
sunt conectate la

gram, prin inter-
de intrare sunt

gram, prin inter-
de ieșire sunt



se prin care se

Significația acestor expresii este următoarea:

- ✓ Pentru **citire**. Fluxul de date se creează între tastatură și o variabilă de memorie: **cin** reprezintă tastatura, iar **operatorul de intrare >>** înseamnă transmiterea unei valori de la tastatură.
- ✓ Pentru **scriere**. Fluxul de date se creează între variabila de memorie și monitor: **cout** reprezintă monitorul, iar **operatorul de ieșire <<** înseamnă transmiterea unei valori către monitor.

Rolul operatorilor **>>** și **<<** este de a apela funcții de sistem (funcția care este apelată este determinată de tipul variabilei de memorie) care convertesc:

- ✓ **la intrare**: o secvență de șiruri de caractere într-un tip de dată;
- ✓ **la ieșire**: un tip de dată într-o secvență de șiruri de caractere.

Sintaxa pentru expresiile care creează fluxurile de date este:

Pentru operația de **citire de la tastatură**:

1. Pentru citirea valorii unei singure variabile de memorie identificată prin numele ei:
`cin >> nume_variabilă_memorie;`
2. Pentru citirea valorilor a *n* variabile de memorie identificate prin numele lor:
`cin >> nume_1 >> nume_2 >> ... >> nume_n;`

Observații:

1. Introducerea valorii unei variabile de memorie se termină prin apăsarea tastei **Enter**.
2. Tipul variabilei de memorie nu trebuie să fie un tip utilizator definit prin set de constante. În exemplul următor, operația de citire va produce avertisment la compilare și nu va avea nici un fel de efect asupra variabilei de memorie *x* la execuția programului:

```
typedef enum {DA,NU} boolean;
boolean x;
cin >> x;
```

3. Dacă pentru o variabilă de memorie din listă nu se introduce de la tastatură o valoare care să corespundă tipului ei, operația de citire se blochează (nu se mai citesc valorile nici pentru celelalte variabile de memorie din listă) și se trece la execuția următoarei instrucțiuni din program. Astfel:

- ✓ Pentru tip **întreg** de dată. Dacă se introduce un număr real sau un caracter, operația de citire se blochează și conținutul zonei de memorie nu se modifică. Dacă se introduce un număr întreg care nu aparține domeniului de definiție intern al tipului de dată (de exemplu, un număr întreg mai mare decât valoarea maximă permisă de tip, sau un număr negativ, pentru tipul întreg pozitiv cum este tipul **unsigned**), operația de citire nu se blochează, dar valoarea introdusă va fi codificată conform tipului de dată al variabilei de memorie și nu va mai corespunde celei introduse de la tastatură.
- ✓ Pentru tip **real** de dată. Dacă se introduce un caracter, operația de citire se blochează și conținutul zonei de memorie nu se modifică. Dacă se introduce un număr real cu mai multe cifre la partea fracționară decât permite precizia reprezentării, se vor memora numai atâtea cifre câte permite precizia

reprezentării (de exemplu, dacă se introduce 3.12345678 pentru o dată de tip **float** care are o precizie de 6 cifre, se va memora valoarea 3.123456).

- ✓ Pentru tip **caracter** de dată. Dacă se introduce un șir de caractere în locul unui singur caracter, în zona de memorie a datei se va păstra primul caracter din șir și se va considera că restul caracterelor din șir au fost citite pentru următoarea variabilă de memorie (dacă mai există una în fluxul de citire **cin >>**).

Să considerăm următoarea secvență de instrucțiuni:

```
unsigned a=0; int b=0, c=0;      (1)
cin>>a>>b>>c;                  (2)
```

După secvența de instrucțiuni (1), în memoria internă se va aloca fiecareia dintre cele trei variabile de memorie câte o zonă de 2 octeți, iar conținutul acestor zone de memorie va fi:

Memoria internă a b c

Dacă, în timpul execuției instrucțiunii (2), de la tastatură se introduc valorile: 1 **Enter** 5 **Enter** 10 **Enter**, conținutul acestor zone de memorie va fi:

Memoria internă a b c

iar dacă se introduc valorile -1 **Enter** a **Enter**, operația de citire se blochează după citirea caracterului **a** și se trece la execuția următoarei instrucțiuni, iar conținutul acestor zone de memorie va fi:

Memoria internă a b c

Pentru operația de **scriere pe ecran**:

1. Pentru afișarea valorii unei singure variabile de memorie sau constante simbolice identificate prin numele ei sau pentru afișarea unei singure constante literale:
`cout<< nume_variabilă_memorie|constantă;`
2. Pentru afișarea valorilor a *n* variabile de memorie sau constante simbolice identificate prin numele lor sau constante literale precizate prin valorile lor:
`cout<< nume_1|const_1<<nume_2|const_2<<...<<nume_n|const_n;`

Observație: Valorile precizate în listă se afișează unele după altele, fără spațiu între ele. De exemplu, în urma execuției instrucțiunilor:

```
int a=10, b=20;
cout<<a<<b;
```

se va afișa 1020. Pentru a evidenția fiecare valoare afișată se pot folosi două metode:

- ✓ valorile se scriu pe același rând, separate prin spații (folosindu-se o constantă de tip șir de caractere pentru afișarea spațiilor):

```
cout<<a<<" "<<b;      se afișează: 10 20
```

- ✓ valorile se scriu pe rânduri diferite, pentru separarea lor folosindu-se constanta **endl** care este o constantă din bibliotecile sistemului și care corespunde apăsării tastei **Enter**:

```
cout<<a<<endl<<b;      se afișează: 10
20
```



Folosirea acestei constante este echivalentă cu folosirea constantelor **linie nouă** și **retur de car**, următoarea instrucțiune fiind echivalentă cu cea anterioară:

```
cout<<a<<"\n"<<"\r"<<b;           se afișează: 10
                                     20
```

Observație: Dacă se afișează o constantă de tip șir de caractere în care vreți să afișați și caracterul ghilimele care este folosit ca delimitator, veți folosi secvența escape pentru acest caracter. De exemplu, în urma execuției instrucțiunii:

```
cout<<" Raspunsul este \"Da \"";
```

pe ecran se va afișa: Raspunsul este "Da".



Limbajul C++ folosește foarte puține instrucțiuni și foarte multe funcții de sistem (cum sunt cele apelate prin expresiile **cin>>** și **cout<<**) care sunt definite în fișierele bibliotecii ale limbajului. Atunci când întâlnește în program o instrucțiune în

care este apelată o funcție de sistem, compilatorul, pentru a putea traduce programul sursă, are nevoie de informații despre acea funcție: numele funcției, numărul și tipul parametrilor folosiți pentru apel și tipul rezultatului furnizat. Aceste informații se numesc **prototipul** funcției. Prototipurile funcțiilor de sistem se găsesc în fișiere antet (header) care au extensia **.h**. Constanta **endl** este o constantă simbolică de sistem. Și în acest caz compilatorul are nevoie de informații: numele constantei, tipul și valoarea. Aceste informații se găsesc tot într-un fișier antet. Pentru a putea fi folosite de compilator, informațiile despre funcțiile sistem apelate (prototipurile lor) și despre constantele simbolice de sistem trebuie incluse în fișierul sursă (fișierul care conține programul pe care l-ați scris) înainte de a începe compilarea lui. Această operație se face scriind în fișierul sursă, înainte de funcția **main()**, directiva pentru preprocesor:

```
#include <nume_fișier_antet.h>
```

Preprocesarea este operația prin care se furnizează informații suplimentare compilatorului. Ea se execută înaintea compilării propriu-zise, prin intermediul unui program special numit **preprocesor**, care acționează asupra programului sursă (fișier text cu extensia **cpp**), iar rezultatul este tot un program sursă (fișier text cu extensia **cpp**), spre deosebire de compilare, care acționează asupra programului sursă (fișier text cu extensia **cpp**), iar rezultatul este un program obiect (fișier cu extensia **obj**). Informațiile suplimentare sunt furnizate compilatorului prin intermediul **directivelor de preprocesare**. Acestea sunt instrucțiuni destinate compilatorului, pe care el trebuie să le execute înainte de a începe compilarea propriu-zisă a programului sursă. Ele formează un **limbaj în interiorul limbajului C**. Directivele de preprocesare încep cu caracterul **#** care este urmat de **numele directivei** și nu se termină cu caracterul **;**.

Astfel, prin directiva de preprocesare **#include <nume_fișier_antet.h>** în fișierul sursă va fi inclus fișierul antet care conține prototipuri ale funcțiilor de sistem.

Un fișier antet conține prototipurile mai multor funcții de sistem. Prin includerea fișierului antet în fișierul sursă, vor fi incluse toate prototipurile de funcții din acel fișier, chiar dacă nu apelați toate acele funcții în program. Există mai multe fișiere antet și este posibil ca funcțiile apelate în program să aibă prototipul în fișiere diferite. În acest caz va trebui să includeți mai multe fișiere antet în fișierul sursă, folosind câte

o directivă **#include** pentru fiecare fișier antet. Singura funcție de sistem care nu necesită folosirea unui fișier antet este funcția **main()**.

Prototipurile funcțiilor apelate în fluxurile de citire **cin>>** și de scriere **cout<<** și constanta **endl** se găsesc în fișierul antet **iostream.h**. Orice program în care se creează aceste fluxuri de date trebuie să conțină și directiva de preprocesare:

#include <iostream.h>

De exemplu:

```
#include <iostream.h> //directiva pentru preprocesor
void main()
{int a=1,b; // se declară variabilele a și b
  cout<<"b= "; // se afișează pe ecran constanta de tip sir
                // de caractere b= care este un mesaj de
                // informare */
  cin>>b; // se așteaptă introducerea valorii lui b
  cout<<a<<" "<<b; //se afișează valorile variabilelor a și
  cout<<a<<endl<<b; //b separate prin spații și apoi pe
                    // rânduri diferite
}
```

Evaluare

Răspundeți:

1. Cu ce caracter trebuie să se termine o instrucțiune în limbajul C++?
2. Dați exemple de patru identificatori corecți și patru exemple de identificatori care nu sunt corecți.
3. Ce cuvinte cheie puteți folosi ca nume de variabile de memorie?
4. Care este diferența dintre o constantă simbolică și o constantă literală?
5. Cum se declară o constantă simbolică? Dar o constantă literală?
6. Ce cuvinte cheie se folosesc în instrucțiunile declarative? Care este semnificația lor?
7. Care este valoarea următorului număr, exprimat în notație științifică: $7.54E5$?
Dar a numărului $-1.354e-6$?

8. Găsiți greșelile din următoarele instrucțiuni declarative:

```
const int ALFA=50000;    const char OPTIUNE="D";
const float REAL=.00;    const long float AA=-1.7e-3;
unsigned a='A'; float x=-0.;
double y=0xCA; char c=0x6C;
```

9. Unde scrieți în program directiva **#include**?

10. Ce se va afișa pe ecran după executarea următoarei secvențe de instrucțiuni dacă se citește de la tastatură 97?

```
char c; cin>>c; cout<<c;
```

11. Ce se va afișa pe ecran după executarea următoarei secvențe de instrucțiuni dacă se citesc de la tastatură 10 și 1.5?

```
int a=0,b=0; cin>>a>>b; cout<<a<<b;
```

12. Ce se va afișa pe ecran după executarea următoarei secvențe de instrucțiuni dacă se citesc de la tastatură 10, 20 și 30?

```
int a,b; cin>>a>>b>>a; cout<<a<<endl<<b<<endl<<a;
```

#include <iostream>
using namespace std;

Informatică

13. Ce se va afișa pe ecran după executarea următoarei secvențe de instrucțiuni?
`char a1=1, a2=2; cout<<A1<<endl<<A2;`

14. Ce se va afișa pe ecran după executarea următoarei secvențe de instrucțiuni?
`int a=10, b=20, c=30;
cin>>a>>b; cout<<a<<endl<<b<<'\\n'<<'\\r'<<c;`

15. Ce se va afișa pe ecran după executarea următoarei secvențe de instrucțiuni?
`char a=97; int b=a;
cin>>a>>b;`

Adevărat sau fals:

1. Următoarea secvență de instrucțiuni declarative dintr-un program este corectă:
`int a=10, b=a;
float x=b;`

2. Următoarea secvență de instrucțiuni declarative dintr-un program este corectă:
`unsigned char a=98;
double float x=1.5;`

3. Următorul comentariu scris pe mai multe linii este corect:
`/* Acesta este un comentariu
 scris pe mai multe linii`

4. Dacă variabila de memorie *a* este de tip *int*, ea poate avea valoarea 40000.

5. Dacă variabila de memorie *c* este de tip *unsigned char*, ea poate avea valoarea 200.

Alegeți:

1. Care dintre următoarele secvențe reprezintă un program corect?
a) `void main { }` c) `main() { }`
b) `void Main ()` d) `void main()`
`{ }` `( )`

2. Care dintre următoarele secvențe reprezintă un program corect?
a) `main` c) `void main()`
`{ }` `{; }`
b) `void main` d) `void main();`
`{ }` `{ }`

3. Se citește un număr natural care este mai mic de 60000. Tipul de dată pe care-l veți alege pentru variabila de memorie, astfel încât să consumați cât mai puțină memorie internă, este:
a) *long* c) *unsigned long*
b) *unsigned* d) *int*

4. Care dintre următoarele instrucțiuni nu declară o variabilă de memorie reală?
a) `double a=0;` c) `float C;`
b) `long double a;` d) `double float b=3.5;`

5. Care dintre următoarele instrucțiuni nu declară două variabile de memorie de tip întreg?
a) `int a; long b;` c) `long double a; long b;`
b) `unsigned a; long b;` d) `typedef int real; real a,b;`

4.5. Expresia și instrucțiunea expresie

Ați aflat că o expresie este o succesiune de operatori și operanzi legați între ei după reguli specifice limbajului:

- ✓ **Operanzii** pot fi:
 - nume de variabile de memorie sau de constante,
 - constante,
 - funcții care furnizează un rezultat prin chiar numele lor (de exemplu funcția sistem **sqrt(x)**, care furnizează radical de ordinul 2 din x).
- ✓ **Operatorii** sunt simboluri care determină executarea anumitor operații.

Expresia poate să conțină **numai un operand**. De exemplu, constantele literale 11 sau "Mesaj" sunt expresii, iar dacă s-a declarat o variabilă de memorie *a* de tip *int*, *a* este o expresie.

Operatorii pot fi:

- ✓ **unari** – se aplică pe un singur operand,
- ✓ **binari** – se aplică pe doi operanzi,
- ✓ **ternari** – se aplică pe trei operanzi.

În limbajul C++ sunt implementate următoarele tipuri de operatori:

- ✓ operatori aritmetici,
- ✓ operatori de incrementare și decrementare,
- ✓ operatori relaționali,
- ✓ operatori logici,
- ✓ operatori logici pe biți,
- ✓ operatori de atribuire,
- ✓ operatorul virgulă,
- ✓ operatorul condițional,
- ✓ operatorul de conversie explicită,
- ✓ operatorul dimensiune.

Precedența operatorilor. În limbajul C++ există 16 niveluri de prioritate.

Asociativitatea operatorilor. În limbajul C++ există două tipuri de asociativitate: de la stânga la dreapta și de la dreapta la stânga. Operatorii care au același nivel de prioritate au aceeași asociativitate.

Regulă: Toți operatorii unari au același nivel de prioritate, indiferent de tipul lor, prioritatea lor fiind mai mare decât a oricărui operator binar sau ternar, iar asociativitatea lor este de la dreapta la stânga.

În funcție de rezultatul obținut în urma evaluării, există:

- ✓ **Expresii matematice:** rezultatul este un număr întreg sau real.
- ✓ **Expresii logice:** rezultatul este 0 sau 1 care poate fi interpretat ca *False* sau *True*.
- ✓ **Expresii de tip text:** rezultatul este un șir de caractere.
- ✓ **Expresii fără tip,** care conțin apelul unei funcții care nu furnizează nici un rezultat.

În program, expresiile pot fi folosite:

- ✓ ca parametri la apelul unor funcții;
- ✓ în instrucțiunile de control, pentru a stabili modul în care se predă controlul instrucțiunilor din program;
- ✓ în instrucțiunile expresii.

Instrucțiunea expresie are sintaxa:

expresie;

Dacă expresia conține:

- ✓ operatorul de atribuire, se obține o **instrucțiune de atribuire**;
- ✓ numai numele unei funcții care nu furnizează nici un rezultat prin numele ei, se obține o **instrucțiune procedurală**.

4.5.1. Operatorii aritmetici

Operatorii aritmetici implementați în limbajul C++ sunt:

- ✓ **operatori unari:**
 - operatorul minus,
 - + operatorul plus,
- ✓ **operatori binari:**
 - de adunare + operatorul pentru adunare
 - operatorul pentru scădere
 - de multiplicare * operatorul pentru înmulțire
 - / operatorul pentru împărțire
 - % operatorul **modulo** (restul împărțirii)

Operatorii unari preced operandul. Ei se folosesc pentru a stabili semnul unui operand numeric. Astfel, dacă vrei să atribuie valoarea -12 unei variabile de memorie de tip `int` la declararea ei, vei folosi expresia -12 formată din constanta literală 12 (care nu poate să fie decât pozitivă), pe care ai aplicat operatorul unar *minus*.

Precedența operatorilor aritmetici este:

- ✓ Operatorii unari au prioritate mai mare decât operatorii binari.
- ✓ Pentru operatorii binari precedența este cea aritmetică.

Asociativitatea operatorilor aritmetici binari este de la stânga la dreapta.

Observații:

1. Operatorul / se folosește și pentru împărțirea întreagă (semnificația operatorului `div` prezentat la algoritmi), dar și pentru împărțirea reală. Dacă *a* și *b* sunt operanzii pe care se aplică operatorul, iar *c* este rezultatul ($c \leftarrow a/b$), semnificația operatorului **depinde de tipul operanzilor**, astfel:

- ✓ Dacă ambii operanzi *a* și *b* sunt de tip întreg, rezultatul *c* va fi de tip întreg, iar semnificația operatorului va fi de împărțire întreagă.
- ✓ Dacă cel puțin unul dintre operanzii *a* și *b* este de tip real, rezultatul *c* va fi de tip real, iar semnificația operatorului va fi de împărțire reală.

Exemplu:

```
#include <iostream.h>
void main()
{int a=10;          // se declară variabila a de tip întreg
```



```

float b=10;      // se declară variabila b de tip real
cout<<a/3;      /* se afișează 3 (rezultat de tip întreg)
                 care reprezintă câtul dintre a și 3;
                 s-a efectuat împărțirea întreagă. */
cout<<b/3;      /* se afișează 3.333333 (rezultat de tip
                 real); s-a efectuat împărțirea reală. */
cout<<a/3.;     /* se afișează 3.333333 (rezultat de tip
                 real); s-a efectuat împărțirea reală
                 deoarece împărțitorul este real (o
                 constantă reală) */

```

2. Operatorul % nu se poate aplica pe valori numerice reale (pe operanzi de tipul float sau double).
3. Operatorii / și % se pot folosi pentru a calcula câtul (c) și restul (r) împărțirii a două numere întregi a și b cu semn. Dacă notăm cu |a| modul de a, cu |b| modul de b, cu c câtul dintre |a| și |b| ($c = |a|/|b|$) și cu r restul împărțirii lui |a| la |b| ($r = |a| \% |b|$), atunci semnul câtului și al restului împărțirii lui a la b va fi cel din tabelul alăturat.

a	b	a/b	a%b
>0	>0	c	r
>0	<0	-c	r
<0	>0	-c	-r
<0	<0	c	-r

Exemplu:

```

#include <iostream.h>
void main()
{int a=5, b=2;
  cout<<a/b<<" "<<a%b<<endl;      // se afișează 2 1
  cout<<-a/b<<" "<<-a%b<<endl;     // se afișează -2 -1
  cout<<a/-b<<" "<<a%-b<<endl;     // se afișează -2 1
  cout<<-a/-b<<" "<<-a%-b<<endl;   // se afișează 2 -1 }

```

4. Pentru a schimba ordinea implicită de executare a operatorilor matematici dată de prioritatea lor, se pot folosi parantezele rotunde () care au rol de separatori.

Exemplu:

```

#include <iostream.h>
void main()
{int a=10, b=20;
  cout<<a+b*2;      // se afișează 50
  cout<<(a+b)*2;    // se afișează 60 }

```

5. O expresie aritmetică poate conține operanzi numerici de mai multe tipuri. De exemplu, dacă a și b sunt operanzi numerici de două tipuri diferite (char și int sau int și float), iar c este rezultatul, tipul rezultatului c se va obține prin **conversie aritmetică implicită**. Pentru a nu se pierde informația, această conversie se face înainte de executarea operației și funcționează prin „promovarea” tipului inferior, adică a tipului „mai puțin încăpător” la tipul superior, adică tipul „mai încăpător”, rezultatul având tipul superior. Regulile de conversie aritmetică implicită sunt:

- Pas 1.** Dacă unul dintre operanzi este de tip long double, și celălalt este convertit în tipul long double; altfel, se trece la Pas 2.

- Pas 2.** Dacă unul dintre operanzi este de tip **double**, și celălalt este convertit în tipul **double**; altfel, se trece la **Pas 3**.
- Pas 3.** Dacă unul dintre operanzi este de tip **float**, și celălalt este convertit în tipul **float**; altfel, se trece la **Pas 4**.
- Pas 4.** Dacă unul dintre operanzi este de tip **unsigned long**, și celălalt este convertit în tipul **unsigned long**; altfel, se trece la **Pas 5**.
- Pas 5.** Dacă unul dintre operanzi este de tip **long**, și celălalt este convertit în tipul **long**; altfel, se trece la **Pas 6**.
- Pas 6.** Convertește tipurile **char**, **unsigned char** și **short** în tipul **int**.

Exemplul 1:

```
#include <iostream.h>
void main()
{float a=1;      /* se declară variabila a de tip float și se
                  inițializează cu valoarea 1 */
  char b='a'     /* se declară variabila b de tip char și se
                  inițializează cu constanta de tip caracte-
                  ter 'a' care are codul ASCII 97 */
  cout<<a+b;     // se afișează 98 (rezultat de tip float)}
```

Conversia implicită s-a efectuat astfel: tipul inferior **char** a fost „promovat” la tipul superior **float**, iar rezultatul este de tip **float**.

Exemplul 2:

```
#include <iostream.h>
void main()
{const float PI=3.14; // se declară constanta PI de tip float
  int raza=2;        // se declară variabila raza de tip int
  cout<<"aria= "<<PI*raza*raza; // se afișează aria= 12.56
  /* se afișează o constantă de tip sir de caractere și
   rezultatul unei expresii aritmetice; rezultatul este de
   tip float */ }
```

La evaluarea expresiei, conversia implicită s-a efectuat astfel: tipul inferior **int** (al variabilei *raza*) a fost „promovat” la tipul superior **float** (al constantei simbolice *PI*), iar rezultatul este de tip **float**.

4.5.2. Operatorul pentru conversie explicită

Conversia de tip poate fi forțată să se execute în alt mod decât cel implicit, folosind operatorul de conversie explicită care mai este numit și operatorul **typecast**. Este un operator unar care precede operandul. Sintaxa sa este:

(tip_dată)

unde **tip_dată** este tipul datei în care se convertește operandul. De exemplu, operatorul (**int**) convertește operandul în tipul **int**, iar operatorul (**float**) convertește operandul în tipul **float**.

Observație: Operatorul de conversie explicită de tip nu modifică tipul variabilei de memorie pe care este aplicat. Conversia se face numai în cadrul operației de evaluare a expresiei.

- ✓ fie folosiți pentru unul dintre operanzi o constantă reală,
- ✓ fie convertiți tipul unui operand într-un tip real, cu ajutorul operatorului de conversie explicită (**float**).

4.5.3. Operatorii pentru incrementare și decrementare

Sunt operatori unari care se aplică pe un operand numeric:

- ✓ **Operatorul de incrementare ++** adună 1 la valoarea operandului. **a++** corespunde operației de atribuire din pseudocod: $a \leftarrow a+1$.
- ✓ **Operatorul de decrementare --** scade 1 din valoarea operandului. **a--** corespunde operației de atribuire din pseudocod: $a \leftarrow a-1$.

Acești operatori pot fi:

- ✓ **Prefixați**, adică aplicați înaintea operandului: **++a** sau **--a**. În acest caz incrementarea, respectiv decrementarea operandului se face înainte ca valoarea operandului să intre în calcul (înainte să fie evaluată expresia).
- ✓ **Postfixați**, adică aplicați după operand: **a++** sau **a--**. În acest caz incrementarea, respectiv decrementarea operandului se face după ce valoarea operandului a intrat în calcul (după ce a fost evaluată expresia).

Exemplul 1:

```
#include <iostream.h>
void main()
{
    int a=1,b=2;
    cout<<a+++b++<<endl;           //se afișează 3
    cout<<a<<" "<<b<<endl;         //se afișează 2 3
    cout<<++a+(++b)<<endl;          //se afișează 7
    cout<<a<<" "<<b<<endl;         //se afișează 3 4
    cout<<a+b++<<endl;             //se afișează 7
    cout<<a<<" "<<b<<endl;         //se afișează 3 5
    cout<<a+++b--<<endl;           //se afișează 8
    cout<<a<<" "<<b<<endl;         //se afișează 4 4
    cout<<a+++--b<<endl;           //se afișează 7
    cout<<a<<" "<<b<<endl;         //se afișează 5 3
    cout<<++a+b++<<endl;          //se afișează 9
    cout<<a<<" "<<b<<endl;         //se afișează 6 4 }
}
```

Operatorii de incrementare se pot aplica pe orice tip numeric de dată.

Exemplul 2:

```
#include <iostream.h>
void main()
{
    float a=1.5,b=-2.5;
    char x='a';
    cout<<a+++b++<<endl;           //se afișează -1
    cout<<a<<" "<<b<<endl;         //se afișează 2.5 -1.5
    cout<<a--b--<<endl;           //se afișează 1
    cout<<a<<" "<<b<<endl;         //se afișează 1.5 -2.5
    cout<<++(++x)<<endl;           //se afișează c
}
```

Operatorii sunt

Observație. În cazul formei postfixate nu este precizat momentul când se face incrementarea atunci când se evaluează expresia. Acest moment depinde de compilator. De exemplu, în următorul caz:

```
int a=2;
cout<< a+++a++;
```

expresia este corectă din punct de vedere sintactic (nu produce eroare de compilare), dar la execuția ei pot să apară două situații, în funcție de modul în care evaluează expresia compilatorul cu care se compilează programul:

- incrementarea se face după evaluarea expresiei și rezultatul expresiei va fi 4, iar valoarea operandului *a* va fi 4;
- incrementarea se face imediat după ce valoarea *a* intrat în calcul: valoarea care intră în calcul pentru primul termen este valoarea inițială a lui *a* (2), după care se incrementează *a*, intră în calcul pentru al doilea termen cu această valoare (3), după care se incrementează din nou *a*, rezultatul expresiei fiind 5, iar valoarea operandului *a*, 4.

Exemplul 3:

```
// Exemplu de evaluare de către compilator a unor expresii
// care conțin incrementări prefixate și postfixate
#include <iostream.h>
void main()
{
    int a=2;
    cout<< ++a+(++a)<<endl;           //se afișează 7
    cout<< a<<endl;                   //se afișează 4
    cout<< (++a)+(++a)<<endl;           //se afișează 11
    cout<< a<<endl;                   //se afișează 6
    cout<< a+++a++<<endl;             //se afișează 13
    cout<< a<<endl;                   //se afișează 8
    cout<< (a++)+a++<<endl;           //se afișează 17
    cout<< a<<endl;                   //se afișează 10
    cout<< ++a+a++<<endl;             //se afișează 22
    cout<< a<<endl;                   //se afișează 12
    cout<< a+++ (++a)<<endl;           //se afișează 26
    cout<< a<<endl;                   //se afișează 14
    cout<< (a++)+(++a)<<endl;           //se afișează 30
    cout<< a<<endl;                   //se afișează 16
    cout<< ((++a)++)+(++a)<<endl;       //se afișează 36
    cout<< a<<endl;                   //se afișează 19
    cout<< (++(++a))+(++a)<<endl;       //se afișează 43
    cout<< a<<endl;                   //se afișează 22
}
```



Evitați folosirea unor expresii a căror evaluare nu este decisă prin reguli precise date de definiția limbajului C++, ci de interpretarea compilatorului, pentru că astfel programul obținut nu este un program portabil. Un **program portabil** este un program care produce aceleași rezultate indiferent de compilatorul C++ cu care a fost compilat.

4.5.4. Operatorii

Toți operatorii

✓ Operatorii

✓ Operatorii

Produc un rezultat

Precedența operatorilor

mare decât precedența

operatorii aritmetici

Asociativitatea

asociativitatea

Exemplu:

```
#include <iostream.h>
void main()
{
    int a=2;
    cout<< a<<endl;
    cout<< (a++)<<endl;
    cout<< (a++)<<endl;
    cout<< (a++)<<endl;
    cout<< (a++)<<endl;
}
```



și evaluarea

evaluată corect

4.5.5. Operatorii

Există trei operatori

✓ operatorii

✓ operatorii

Se aplică pe

și orice valoare

pentru fals și

Precedența

✓ Operatorii

✓ Pentru operatorii

Asociativitatea

stânga la dreapta

Operatorii logici

4.5.4. Operatorii relaționali

Toți operatorii relaționali sunt operatori binari. Ei se împart în două grupe:

- ✓ Operatorii relaționali pentru inegalități: $<$, $>$, $<=$ și $>=$.
- ✓ Operatorii relaționali pentru egalitate $==$ (egal) și $!=$ (diferit).

Produc un rezultat numeric: 0 pentru *fals* și 1 pentru *adevărat*.

Precedența operatorilor relaționali este: operatorii din prima grupă au prioritate mai mare decât cei din a doua grupă. Operatorii relaționali au prioritate mai mică decât operatorii aritmetici.

Asociativitatea operatorilor relaționali este: pentru aceeași grupă de prioritate, asociativitatea este de la stânga la dreapta.

Exemplu:

```
#include <iostream.h>
void main()
{int a=2,b=3;
cout<<(a>b)<<endl;           //se afișează 0
cout<<(a<=b)<<endl;          //se afișează 1
cout<<(a==b)<<endl;          //se afișează 0
cout<<(a!=b);                //se afișează 1
}
```



Cu fluxul **cout<<** ați afișat rezultatul unei expresii matematice și puteți afișa și rezultatul unei expresii logice. Operatorul de ieșire **<<** are prioritate mai mică decât operatorii matematici și unari, dar mai mare decât a operatorilor relaționali și logici. Deoarece **cout<<** are prioritate

și evaluarea se face de la stânga la dreapta, pentru ca întreaga expresie să fie evaluată corect, scrieți expresia logică între paranteze rotunde.

4.5.5. Operatorii logici

Există trei operatori logici:

- ✓ operatorul unar: **!** operatorul pentru negație logică,
- ✓ operatorii binari: **&&** operatorul **ȘI** logic, **||** operatorul **SAU** logic.

Se aplică pe orice variabilă sau constantă de tip numeric (valoarea 0 înseamnă *fals* și orice valoare diferită de 0 înseamnă *adevărat*) și produc un rezultat numeric: 0 pentru *fals* și 1 pentru *adevărat*.

Precedența operatorilor logici este:

- ✓ Operatorul unar are prioritate mai mare decât operatorii binari.
- ✓ Pentru operatorii binari operatorul **&&** are prioritate mai mare decât operatorul **||**.

Asociativitatea operatorilor logici binari cu același nivel de prioritate este de la stânga la dreapta.

Operatorii logici au prioritate mai mică decât operatorii relaționali.

Exemplu:

```
#include <iostream.h>
void main()
{float a=2.5;
int b=0,c=2,d=3;
cout<<(a>0)<<endl;           //se afișează 1
cout<<!a<<endl;              //se afișează 0
cout<<(c&&d)<<endl;           //se afișează 1
cout<<(b&&c)<<endl;           //se afișează 0
cout<<(b||c);                 //se afișează 1
cout<<(a&&d)<<endl;           //se afișează 1
cout<<(b&&a)<<endl;           //se afișează 0
cout<<(b||a);                 //se afișează 1 }
```

Atenție

În cazul operatorilor logici binari al doilea operand nu mai este evaluat, dacă prin evaluarea primului operand se poate decide valoarea rezultatului:

- ✓ Operatorul **&&**. Dacă primul operand are valoarea 0, rezultatul este 0 oricare ar fi valoarea celui de al doilea operand. Dacă primul operand are valoarea 1, rezultatul depinde de al doilea operand. În acest caz, nu se mai evaluează al doilea operand, dacă primul operand are valoarea 0.
- ✓ Operatorul **||**. Dacă primul operand are valoarea 1, rezultatul este 1 oricare ar fi valoarea celui de al doilea operand, iar dacă primul operand are valoarea 0, rezultatul depinde de al doilea operand. În acest caz, nu se mai evaluează al doilea operand, dacă primul operand are valoarea 1.

Exemplu:

```
#include <iostream.h>
void main()
{int a=0,b=1;
cout<<(a&&b++)<<endl;         //se afișează 0
cout<<b<<endl;                 //se afișează 1
cout<<(b||a++)<<endl;         //se afișează 1
cout<<a;                       //se afișează 0
}
```

4.5.6. Operatorii logici pe biți

Acționează asupra operanzilor de tip întreg și execută operații logice la nivel de biți. Există următorii operatori logici pe biți:

- ✓ **operatorul unar:** ~ operatorul negare pe biți,
- ✓ **operatori binari:**
 - >> operatorul pentru deplasare la dreapta
 - << operatorul pentru deplasare la stânga
 - & operatorul pentru ȘI pe biți
 - ^ operatorul pentru SAU exclusiv pe biți
 - ! operatorul pentru SAU pe biți

- Precedența operatorilor logici pe biți este:

- ✓ Operatorul unar are prioritate mai mare decât operatorii binari.
- ✓ Operatorii de deplasare au prioritate mai mare decât ceilalți operatori binari logici pe biți.

- ✓ Operatorul ȘI pe biți (&) are prioritate mai mare decât operatorul SAU exclusiv pe biți (^) care are prioritate mai mare decât operatorul SAU pe biți (|).

Asociativitatea operatorilor logici pe biți binari cu același nivel de prioritate este de la stânga la dreapta.

Operatorii de deplasare au prioritate mai mică decât operatorii aritmetici, dar mai mare decât operatorii relaționali. Ceilalți operatori binari logici pe biți au prioritate mai mică decât operatorii relaționali, dar mai mare decât operatorii logici.

Operatorii de deplasare

Operatorul de deplasare la stânga $a \ll n$ deplasează biții operandului a cu n poziții la stânga. Primii n biți se pierd, iar cele n poziții rămase libere în dreapta sunt completate cu 0.

Operatorul de deplasare la dreapta $a \gg n$ deplasează biții operandului a cu n poziții la dreapta. Ultimii n biți se pierd, iar cele n poziții rămase libere în stânga sunt completate cu 0.

Observații:

1. Deplasarea a n biți la dreapta în operandul a este echivalentă cu câtul împărțirii operandului a la 2^n .
2. Deplasarea a n biți la stânga în operandul a este echivalentă cu înmulțirea operandului a cu 2^n .



Operatorul de deplasare la stânga pe bit și operatorul de ieșire pentru fluxul de date **cout** folosesc aceeași reprezentare simbolică: $\ll$. Deoarece evaluarea se face de la stânga la dreapta, operatorul de ieșire are prioritate și, pentru a afișa rezultatul operației de deplasare la stânga $a \ll n$, trebuie să

încadrați expresia între paranteze rotunde: corect este **cout<<(a<<n)**, și nu **cout<<a<<n**. Astfel, **cout<<(16<<1)**; afișează 32, iar **cout<<16<<1**; afișează 161.

Exemplul 1:

```
#include <iostream.h>
void main()
{int a=60,n=3;
 cout<<(a>>n)<<endl; //se afișează 7
}
```

Deplasarea operandului a (care are valoarea 60) cu n biți la dreapta (n fiind egal cu 3), este echivalentă cu câtul împărțirii operandului a la 2^n (câtul împărțirii lui 60 la 8). Dacă se face conversia operandului a din baza 10 în baza 16 și în baza 2 se obține: $60_{(10)} = 003C_{(16)} = 00\dots000111100_{(2)}$. După deplasarea cu 3 biți la dreapta, ultimii trei biți se pierd, pozițiile rămase libere în stânga sunt completate cu 0 și valoarea operandului a este: $00\dots0000111_{(2)} = 7_{(16)} = 7_{(10)}$.

a înainte de deplasare										a după deplasare									
0	...	0	1	1	1	1	0	0		0	...	0	0	0	0	1	1	1	

$a \gg n$

Exemplul 2:

```
#include <iostream.h>
void main()
{int a=15,n=3;
  cout<<(a<<n);    //se afișează 120
}
```

Deplasarea operandului a (care are valoarea 15) cu n biți la dreapta (n fiind egal cu 3), este echivalentă cu înmulțirea operandului a cu 2^n (înmulțirea lui 15 cu 8). Dacă se face conversia operandului a din baza 10 în baza 16 și în baza 2 se obține: $15_{(10)} = 000F_{(16)} = 00\dots00001111_{(2)}$. După deplasarea cu 3 biți la stânga, primii trei biți se pierd, pozițiile rămase libere în dreapta sunt completate cu 0 și valoarea operandului a este: $00\dots00001111000_{(2)} = 0078_{(16)} = 120_{(10)}$.

a înainte de deplasare

0	...	0	0	0	1	1	1	1
---	-----	---	---	---	---	---	---	---

 $a \ll n$

a după deplasare

0	...	1	1	1	1	0	0	0
---	-----	---	---	---	---	---	---	---

Operatorul pentru negare pe biți

Operatorul pentru negare pe biți $\sim$ a neagă fiecare bit al operandului a , astfel: $\sim 1 = 0$ și $\sim 0 = 1$.

Exemplu:

```
#include <iostream.h>
void main()
{unsigned int a=0xF2F,b=0x5;
  /* inițializarea s-a făcut cu constante hexazecimale, iar
     variabilele a și b sunt întregi pozitive */
  cout<<(~a)<<" "<<(~b);    // se afișează 61648 65530
}
```

Dacă se face conversia operandului b din baza 16 în baza 2 se obține: $05_{(16)} = 00\dots000101_{(2)}$. După negarea pe biți valoarea operandului b este: $11\dots111010_{(2)} = FFFA_{(16)} = 65530_{(10)}$. Verificați modul în care s-a făcut negarea pe biți a operandului a .

b înainte de negarea pe biți

0	...	0	0	0	0	1	0	1
---	-----	---	---	---	---	---	---	---

 $\sim b$

b după negarea pe biți

1	...	1	1	1	1	0	1	0
---	-----	---	---	---	---	---	---	---

Operatorul ȘI pe biți

Operatorul ȘI pe biți $a \& b$ aplică operatorul logic ȘI pe perechile de biți de pe aceeași poziție din cei doi operanzi a și b : dacă ambii biți sunt 1, rezultatul este 1, altfel este 0.

Exemplu:

```
#include <iostream.h>
void main()
{unsigned int a=0xF2F,b=0x5;
  cout<<(a&b);    //se afișează 5
}
```

Dacă se face conversia operandului a din baza 16 în baza 2 se obține: $0F2F_{(16)} = 00\dots0111100101111_{(2)}$, iar dacă se face conversia operandului b din baza 16 în

baza 2 se
cei doi ope

Operator

Operatorul
perechile c
aceeași va

Exemplu:

```
#include
void mai
{unsigne
cout<<(a
}
```

Dacă se f
00...0111
baza 2 se
cei doi op

Operator

Operator
aceeași p
este 0, al

Exemplu

```
#include
void ma
{unsign
cout<<
}
```

Dacă se
00...0111
baza 2 s
cei doi o

Informatică

baza 2 se obține: $05_{(16)} = 00...000101_{(2)}$. După aplicarea operatorului $\&$ pe biți pe cei doi operanzi, rezultatul este: $00...0000101_{(2)} = 05_{(16)} = 5_{(10)}$.

a	0	...	0	0	1	1	1	1	1	0	0	1	0	1	1	1	1
b	0	...	0	0	0	0	0	0	0	0	0	0	0	1	0	1	1
a & b	0	...	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1

Operatorul SAU exclusiv pe biți

Operatorul SAU exclusiv pe biți $a \wedge b$ aplică operatorul logic SAU exclusiv pe perechile de biți de pe aceeași poziție din cei doi operanzi a și b : dacă ambii biți au aceeași valoare, rezultatul este 0, altfel este 1.

Exemplu:

```
#include <iostream.h>
void main()
{unsigned int a=0xF2F,b=0x5;
cout<<(a^b); //se afișează 3882
}
```

Dacă se face conversia operandului a din baza 16 în baza 2 se obține: $0F2F_{(16)} = 00...011110010111_{(2)}$, iar dacă se face conversia operandului b din baza 16 în baza 2 se obține: $05_{(16)} = 00...000101_{(2)}$. După aplicarea operatorului $\wedge$ pe biți pe cei doi operanzi, rezultatul este: $00...0001111001010_{(2)} = 0F2A_{(16)} = 3882_{(10)}$.

a	0	...	0	0	0	1	1	1	1	0	0	1	0	1	1	1	1
b	0	...	0	0	0	0	0	0	0	0	0	0	0	1	0	1	1
a ^ b	0	...	0	0	0	1	1	1	1	0	0	1	0	1	0	1	0

Operatorul SAU pe biți

Operatorul SAU pe biți $a | b$ aplică operatorul logic SAU pe perechile de biți de pe aceeași poziție din cei doi operanzi a și b : dacă ambii biți au valoarea 0, rezultatul este 0, altfel este 1.

Exemplu:

```
#include <iostream.h>
void main()
{unsigned int a=0xF2F,b=0x5;
cout<<(a|b); //se afișează 3887
}
```

Dacă se face conversia operandului a din baza 16 în baza 2 se obține: $0F2F_{(16)} = 00...011110010111_{(2)}$, iar dacă se face conversia operandului b din baza 16 în baza 2 se obține: $05_{(16)} = 00...000101_{(2)}$. După aplicarea operatorului $|$ pe biți pe cei doi operanzi, rezultatul este: $00...00011110010111_{(2)} = 0F2F_{(16)} = 3887_{(10)}$.

a	0	...	0	0	0	1	1	1	1	0	0	1	0	1	1	1	1
b	0	...	0	0	0	0	0	0	0	0	0	0	0	1	0	1	1
a b	0	...	0	0	0	1	1	1	1	0	0	1	0	1	1	1	1

4.5.7. Operatorii de atribuire

Operatorul de atribuire = este un operator binar ($a=b$) care atribuie primului operand (a) valoarea celui de al doilea operand (b). Construcția $a=b$ este o expresie, rezultatul evaluării ei fiind valoarea primului operand (a).

Observație. Și în cazul operatorului de atribuire acționează conversia implicită de tip, tipul expresiei $a=b$ fiind dat de tipul primului operand (a).

Există trei moduri în care puteți folosi acest operator:

- ✓ atribuirea simplă;
- ✓ atribuirea multiplă;
- ✓ atribuirea cu operator.

Operatorul de atribuire are prioritate mai mică decât toți operatorii prezenți.

Atribuirea simplă

Primul operand este un nume de variabilă de memorie, iar al doilea operand este o expresie (poate fi și numai un nume de variabilă de memorie sau de constantă simbolică sau o constantă literală):

nume_variabilă = expresie

Operația de atribuire simplă se execută în două etape:

- ✓ Se evaluează expresia.
- ✓ Valoarea obținută se atribuie variabilei de memorie, făcându-se, dacă este cazul, conversia de tip.

Exemplul 1:

```
#include <iostream.h>
void main()
{unsigned int x=5,y=2;
 float a,b,c,d;
 a=5*x/y;
 cout<<(a=5*x/y)<<endl;
 //se afișează 12 - rezultatul expresiei
 b=5*(x/y); c=5.*x/y; d=5.*(x/y);
 cout<<a<<" "<<" "<<b<<" "<<c<<" "<<d<<endl;
 // se afișează 12 10 12.5 10
 a=5*(float)x/y; b=5*((float)x/y);
 cout<<a<<" "<<" "<<b; //se afișează 12.5 12.5
}
```

Observație. Și în cazul operatorului de atribuire acționează conversia implicită de tip, tipul expresiei $a=b$ fiind dat de tipul primului operand (a). Dacă tipul primului operand este inferior tipului celui de al doilea operand, pot să apară pierderi de date.

Exemplul 2:

```
#include <iostream.h>
void main()
{unsigned int a=2000;
 char c='a';
 a=c;
```

```

cout<<a<<endl;           //se afișează 97
a=20; c=a;
cout<<(unsigned int)c<<endl; //se afișează 20
a=2000; c=a; a=c;
cout<<(unsigned int)c<<a;   //se afișează 65488 65488
}

```

Observație. Dacă unei variabile de tip întreg i se atribuie valoarea unei variabile reale, valoarea acesteia este **trunchiată**.

Exemplul 3:

```

#include <iostream.h>
void main()
{int a;
 float b=1.8, c=-2.7;
 a=b;
 cout<<a<<endl;           //se afișează 1
 a=c;
 cout<<a<<endl;           //se afișează -2
}

```

Exemplul 4:

```

/*Programul citește un număr real subunitar care are trei
cifre în partea zecimală. Să se afișeze numărul obținut prin
eliminarea primei cifre zecimale și a ultimei cifre zeci-
male (de exemplu, se citește 0.123 și se afișează 0.2).*/
#include <iostream.h>
void main()
{float x;
 cout<<"x= "; cin>>x;
 x=x*10;           //x=1.23
 x=(x-(int)x)*10;   //x=(1.23-1)*10=2.3
 x=((int)x)/10.;    //x=2/10.=0.2
 cout<<x;
}

```

Folosind operatorul de atribuire simplă, se pot interschimba valorile a două variabile de memorie.

Exemplul 5:

```

#include <iostream.h>
void main()
{int a,b,x;
 cout<<"a="; cin>>a;
 cout<<"b="; cin>>b;
 x=a; a=b; b=x;
 cout<<a<<" "<<b;
}

```

Atribuirea multiplă

Deoarece operația de atribuire simplă este o expresie, rezultatul ei poate fi atribuit unei alte variabile de memorie, rezultând o nouă expresie, al cărei rezultat poate fi atribuit unei alte variabile de memorie ș.a.m.d., rezultând o **atribuire multiplă**, prin

care aceeași valoare furnizată de o expresie se atribuie mai multor variabile de memorie identificate prin **nume_1**, **nume_2**, ..., **nume_n**:

nume_1 = nume_2 = ... = nume_n = expresie

Operația de atribuire multiplă se execută în mai multe etape:

- ✓ Se evaluează expresia.
- ✓ Valoarea obținută se atribuie variabilei de memorie identificată cu **nume_n**, făcându-se, dacă este cazul, conversia de tip.
- ✓ Conținutul variabilei **nume_n** se atribuie variabilei de memorie identificate cu **nume_n_1**, făcându-se, dacă este cazul, conversia de tip.
- ✓
- ✓ Conținutul variabilei **nume_2** se atribuie variabilei de memorie identificate cu **nume_1**, făcându-se, dacă este cazul, conversia de tip.

Asociativitatea operatorilor de atribuire este de la dreapta la stânga.

Din această cauză, o expresie care folosește atribuirea multiplă, de genul:

nume_1 = nume_2 = ... = nume_n = expresie_1 = expresie_2

este greșită, deoarece rezultatul unei expresii nu se poate atribui altei expresii.

Exemplu:

```
#include <iostream.h>
void main()
{unsigned int x,y;
float a;
a=x=y=5/2;
cout<<a<<" "<<x<<" "<<y<<endl; //se afișează 2 2 2
a=x=y=5./2;
cout<<a<<" "<<x<<" "<<y<<endl; //se afișează 2 2 2
x=y=a=5./2;
cout<<a<<" "<<x<<" "<<y<<endl; //se afișează 2.5 2 2
x=a=y=5./2;
cout<<a<<" "<<x<<" "<<y<<endl; //se afișează 2 2 2
}
```

Atribuirea cu operator

Primul operand este un nume de variabilă de memorie, iar al doilea operand este o expresie:

nume_variabilă operator = expresie

Operatorul poate fi:

- ✓ un operator aritmetic, și operația se numește **atribuire aritmetică** (**+=**, **-=**, ***=** sau **/=**), sau
- ✓ un operator logic pe biți, și operația se numește **atribuire logică pe biți** (**<<=**, **>>=**, **&=**, **^=** sau **|=**).

Operația de atribuire cu operator se execută în trei etape:

- ✓ Se evaluează expresia.
- ✓ Se aplică operatorul, astfel: primul operand este numele variabilei de memorie, iar al doilea operand este valoarea expresiei. Dacă este cazul, se face conversia de tip.

- ✓ Valoarea obținută se atribuie variabilei de memorie, făcându-se, dacă este cazul, conversia de tip.

De exemplu, operația de atribuire cu operator **a+=5** este echivalentă cu operația de atribuire simplă: **a=a+5**, operația **a&=b** este echivalentă cu operația **a=a&b**, iar operația **a<<=2** este echivalentă cu operația **a=a<<2**.

Exemplul 1:

```
#include <iostream.h>
void main()
{int a=10,b=20,c=30;
  a+=b+c;           // a=a+b+c
  cout<<a;           //se afișează 60
}
```

Observație. Se pot folosi atribuiri multiple cu operatori:

Exemplul 2:

```
#include <iostream.h>
void main()
{int a=10,b=20,c=10;
  a+=b+=c;           //b=b+c și a+=b+c este echivalent cu a=a+b+c;
  cout<<a<<" "<<b;   //se afișează 40 30
}
```

4.5.8. Operatorul condițional

Operatorul condițional **?:** este singurul operator ternar:

expresie_1 ? expresie_2 : expresie_3

Executarea acestui operator se face astfel:

- ✓ Se evaluează **expresie_1**.
- ✓ Dacă valoarea obținută este diferită de 0 (valoarea logică *True*), atunci se evaluează **expresie_2**, iar **expresie_3** va fi ignorată. Valoarea furnizată de operator va fi valoarea expresiei **expresie_2**.
- ✓ Dacă valoarea obținută este 0 (valoarea logică *False*), atunci se evaluează **expresie_3**, iar **expresie_2** va fi ignorată. Valoarea furnizată de operator va fi valoarea expresiei **expresie_3**.

Exemplul 1:

```
//Programul afișează modulul unui număr citit de la tastatură
#include <iostream.h>
void main()
{int x;
  cout<<"x="; cin>>x;
  cout<<"modulul = "<<(x>=0?x:-x);}
```

Observație. Tipul rezultatului va fi determinat prin conversia implicită. De exemplu, dacă tipul expresiei **expresie_3** este superior tipului expresiei **expresie_2**, tipul rezultatului este dat de **expresie_3**, chiar dacă se evaluează **expresie_2**.

Exemplul 2:

```
#include <iostream.h>
void main()
{int a=3;
float b=2;
cout<<(a>0?a:b)/2<<endl; // se afișează 1.5
/* rezultatul operatorului condițional este valoarea lui a de
tip float, care este tipul operandului b, chiar dacă nu se
evaluează acest operand */ }
```

Observație. Operatorul condițional poate fi folosit în locul unei structuri de control alternative simple (structura **dacă... atunci... altfel... sfârșit_dacă**).

Exemplul 3:

```
//Programul determină dacă un număr citit de la tastatură
//este un număr par sau un număr impar
#include <iostream.h>
void main()
{int n;
cout<<"n= "; cin>>n;
cout<<(n%2==0?"numar par":"numar impar");}
```

Exemplul 4:

```
//Programul determină dacă valoarea unui număr citit de la
//tastatură este întreagă sau nu
#include <iostream.h>
void main()
{float a;
cout<<"a= "; cin>>a;
cout<<((int)a==a?"este numar intreg":"nu este numar intreg");
}
```

Exemplul 5:

```
/*Programul determină maximul dintre trei numere întregi
citite de la tastatură */
#include <iostream.h>
void main()
{int a,b,c,max;
cout<<"a="; cin>>a;
cout<<"b="; cin>>b;
cout<<"c="; cin>>c;
max=a; // maximul se inițializează cu valoarea lui a
max=max<b?b:max;
/* Dacă max este mai mic decât b, atunci lui max i se
atribuie valoarea lui b */
max=max<c?c:max;
/* Dacă max este mai mic decât c, atunci lui max i se
atribuie valoarea lui c */
cout<<"maxim= "<<max;
}
```

Exemplul 6:

```
//Programul determină dacă un an citit de la tastatură
//este an bisect sau nu
#include <iostream.h>
```

```
void main()
{
    unsigned int n;
    cout<<"anul= "; cin>>n;
    cout<<((n%100!=0 && n%4==0) || (n%400==0))?"anul este bisect":
    "anul nu este bisect");
}
```

În următorul exemplu s-a folosit funcția **sqrt(x)** pentru a calcula radical de ordinul doi din x. Prototipul acestei funcții se găsește în fișierul antet **math.h**:

Exemplul 7:

```
//Programul determină dacă un număr citit de la tastatură
//este pătrat perfect
#include <iostream.h>
#include <math.h>
void main()
{
    unsigned int n;
    cout<<"n= "; cin>>n;
    cout<<(sqrt(n)==(int)sqrt(n))?"pătrat perfect":"nu e patrat
    perfect");
}
```

Observație. Expresia care se evaluează în operatorul condițional poate fi tot un operator condițional, obținându-se **structuri de control alternative simple imbricate**. În următorul exemplu se citește un caracter de la tastatură: dacă este o literă mică, se transformă în literă mare, dacă este o literă mare, se transformă în literă mică, altfel caracterul rămâne neschimbat. Pentru transformare se folosesc codurile ASCII ale caracterelor. Litera mare are codul ASCII mai mic decât cel al literei mici, iar diferența dintre coduri este 32. Astfel, litera A are codul ASCII 65, iar litera a are codul ASCII 97, litera C are codul ASCII 67, iar litera c are codul ASCII 99 ș.a.m.d.

Exemplul 8:

```
//Programul transformă o literă mare în literă mică și o
//literă mică în literă mare
#include <iostream.h>
void main()
{
    char c;
    cout<<"caracterul= "; cin>>c;
    cout<<(c>='a' && c<='z'? (char)(c-32) : (c>='A' && c<='Z'?
    (char)(c+32) : c));
}
```

4.5.9. Operatorul virgulă

Operatorul virgulă , poate fi folosit pentru construirea unei expresii compuse din mai multe expresii:

expresie_1 , expresie_2 , expresie_3 , ... , expresie_n

Executarea acestui operator se face astfel:

- ✓ Se evaluează **expresie_1**.
- ✓ Se evaluează **expresie_2**.
- ✓
- ✓ Se evaluează **expresie_n**.

Rezultatul și tipul rezultatului este dat de valoarea ultimei expresii.

Asociativitatea acestui operator este de la stânga la dreapta. Este operatorul cu cea mai mică prioritate.

Folosirea operatorului virgulă este utilă acolo unde sintaxa nu permite decât o singură expresie și trebuie evaluate mai multe expresii. Expresiile vor fi legate prin operatorul virgulă, obținându-se o singură expresie.

Exemplul 1:

```
#include <iostream.h>
void main()
{int a=5,b=10;
float d,c;
c=a++b, ++a, d=(float)a/b--, ++b;
cout<<a<<" "<<b<<" "<<c<<" "<<d<<endl;
//se afișează 12 11 11 1.090909
a=5,b=10;
cout<<(c=a++b, ++a, c=(float)a/b--, ++b)/2;
//se afișează 5 - rezultatul expresiei este valoarea lui b
//și este de tipul lui b, adică int }
```

Exemplul 2:

```
/*Programul determină maximul dintre trei numere întregi
citite de la tastatură */
#include <iostream.h>
void main()
{int a,b,c,max;
cout<<"a="; cin>>a;
cout<<"b="; cin>>b;
cout<<"c="; cin>>c;
max=a,max=max<b?b:max,max=max<c?c:max;
cout<<"maxim= "<<max;
}
```

Exemplul 3:

```
/*Programul ordonează crescător trei numere întregi
citite de la tastatură */
#include <iostream.h>
void main()
{int a,b,c,max;
cout<<"a="; cin>>a;
cout<<"b="; cin>>b;
cout<<"c="; cin>>c;
a>b?(x=a,a=b,b=x):a;
b>c?(x=b,b=c,c=x):a;
a>b?(x=a,a=b,b=x):a;
cout<<a<<" "<<b<<" "<<c; }
```



Nu confundați caracterul virgulă folosit ca **separator** în liste (lista cu parametrii de apel ai unei funcții, lista cu variabilele dintr-o instrucțiune declarativă etc.) cu **operatorul virgulă**. În cazul operatorului virgulă este garantată evaluarea de la stânga la dreapta, dar în cazul caracterului separator – nu).

4.5.10. Operatorul dimensiune

Operatorul dimensiune **sizeof** este un operator unar. Operandul poate fi o expresie sau un tip de dată:

sizeof(expresie)
sau
sizeof(tip_dată)

Rezultatul furnizat de acest operator reprezintă numărul de octeți utilizați pentru a memora valoarea expresiei sau tipul de dată precizat.

Observație. Aplicarea acestui operator pe o expresie **nu are ca efect și evaluarea expresiei.**

Exemplu:

```
#include <iostream.h>
void main()
{int a=5,b=10;
float c;
cout<<sizeof(int)<<" "<<sizeof(float)<<endl;
/* se afișează 2 4
   (considerând că pe mașina pe care este executat programul
   tipul int ocupă 2 octeți, iar tipul float 4 octeți)*/
cout<<sizeof(c*a++)<<" "<<sizeof(a/b--)<<endl;
/* se afișează 4 2
   rezultatul primei expresii este de tip float, iar
   rezultatul celei de a doua expresii este de tip int */
cout<<a<<" "<<b;
/* se afișează 5 10
   expresiile nu au fost evaluate;
   valorile variabilelor a și b nu au fost modificate */}
```

4.5.11. Precedența și asociativitatea operatorilor

Pe lângă operatorii prezentați, în limbajul C++ mai există și alți operatori. Operatorii prezentați ocupă nivelurile de prioritate astfel:

Nivel	Categorie	Operatori	Semnificație
1	Prioritate maximă	()	Apel de funcție
2	Operatori unari	+ -	Plus și minus
		++ --	Incrementare și decrementare
		!	Negare logică
		~	Negare pe biți
		sizeof	Dimensiune (în octeți)
		(tip)	Conversie explicită de tip
3	Nu au fost prezentați		
4	Operatori aritmetici de multiplicare	* / %	Înmulțire, împărțire și rest (modulo)
5	Operatori aritmetici adiționali	+ -	Adunare și scădere

Nivel	Categorie	Operatori	Semnificație
6	Operatori de deplasare pe biți	<< >>	Deplasare la stânga și deplasare la dreapta
7	Operatori relaționali pentru inegalități	< <= > >=	Mai mic, mai mic sau egal, mai mare și mai mare sau egal
8	Operatori relaționali pentru egalitate	== !=	Egal și diferit
9		&	Și pe biți
10		^	SAU exclusiv pe biți
11			SAU pe biți
12		&&	Și logic
13			SAU logic
14		?:	Operatorul condițional
15	Operatori de atribuire	= += -= *= /= &= ^= = <<= >>=	Atribuire simplă Atribuire cu operatori matematici Atribuire cu operatori logici pe biți
16	Operatorul virgulă	,	Evaluare multiplă

Asociativitatea operatorilor: cu excepția operatorilor unari, condiționali și de atribuire, care au asociativitate de la dreapta la stânga, restul operatorilor au asociativitate de la stânga la dreapta.



Atenție

Greșeli pe care le puteți face atunci când scrieți programul și care vor duce la apariția **erorilor de sintaxă** sau a avertismentelor la compilare:

1. Ați uitat să scrieți caracterul ; la sfârșitul unei instrucțiuni.
2. Apelați în program o funcție standard și ați uitat să includeți fișierul antet al acestei funcții cu directiva pentru preprocesor **#include**.
3. Ați pus caracterul ; la sfârșitul unei instrucțiuni **#include** pentru preprocesor.
4. Ați folosit un identificator greșit de variabilă.
5. Ați folosit un nume de variabilă sau de constantă pe care nu ați declarat-o.
6. Ați uitat să închideți un bloc (nu ați folosit corect parantezele perechi {}).
7. Ați folosit un tip greșit pentru o variabilă de memorie (de exemplu, vreți să atribuiți o constantă caracter unei variabile de tip **int**).
8. Nu ați scris corect un comentariu: fie nu ați închis cu */ un comentariu deschis cu /*, fie ați scris pe mai multe rânduri un comentariu deschis cu /*.
9. Nu ați încadrat între paranteze rotunde () o expresie logică al cărei rezultat îl afișați cu funcția **cout<<**.
10. Ați folosit operatorul de atribuire (=) în locul operatorului relațional egal (==).

Definiție
la stânga și
la dreapta
mic sau egal,
mare sau egal
diferit
pe biți
siv pe biți
pe biți
logic
logic
condițional
simplă
atori matematici
atori logici pe biți
multiplă
condiționali și de atri-
rilor au asociati-
scrieți programul și
ă sau a avertis-
fișierul antet al
preprocesor.
declarat-o.
erechi {}).
exemplu, vreți să
mentariu deschis
cu //.
al cărei rezultat îl
egal (==).

Evaluare

Răspundeți:

1. Care este deosebirea dintre operatorul de incrementare prefixat și operatorul de incrementare postfixat?
2. Ce se afișează în urma executării următoarei secvențe de instrucțiuni, dacă pentru a și b se citește 5? Dar dacă pentru a se citește 7, iar pentru b se citește 5?

```
int a,b;
cout<<"a= "; cin>>a;
cout<<"b= "; cin>>b;
a>b?a-=b:b-=a;
cout<<a<<" "<<b;
```

3. Ce se afișează în urma executării următoarei secvențe de instrucțiuni dacă pentru a și b, se citește 5? Dar dacă pentru a se citește 7, iar pentru b se citește 5?

```
int a,b;
cout<<"a= "; cin>>a;
cout<<"b= "; cin>>b;
a>b?(a--,b++):(b--,a++);
cout<<a<<" "<<b;
```

4. Dacă a1, a2, a3 și a4 sunt patru variabile de memorie de tip int, câte valori poate lua rezultatul expresiei a1 || a2 || a3 || a4? Dar rezultatul expresiei a1 || a2 || a3 || 1?
5. Dacă a1, a2, a3 și a4 sunt patru variabile de memorie de tip int, câte valori poate lua rezultatul expresiei a1 && a2 && a3 && a4? Dar rezultatul expresiei a1 && a2 && a3 && 0?

6. Ce se afișează în urma executării următoarei secvențe de instrucțiuni dacă pentru a și b se citește 5? Dar dacă pentru a se citește 7, iar pentru b se citește 5?

```
int a,b;
cout<<"a= "; cin>>a;
cout<<"b= "; cin>>b;
cout<<(a>b?a-b:b-a);
```

7. Ce se afișează în urma executării următoarei secvențe de instrucțiuni?

```
char c='a';
cout<<++c;
```

8. Ce se afișează în urma executării următoarei secvențe de instrucțiuni dacă pentru a se citește 7, iar pentru b se citește 5?

```
int a,b;
cout<<"a= "; cin>>a;
cout<<"b= "; cin>>b;
a-=b, b+=a, a=b-a;
cout<<a<<" "<<b;
```

9. Ce se afișează în urma executării următoarei secvențe de instrucțiuni dacă pentru a se citește mai întâi 7, iar apoi 5?

```
int a,b,c;
cout<<"a= "; cin>>a; b=a%4;
cout<<"a= "; cin>>a; c=a/2;
cout<<a<<" "<<b<<" "<<a<<" "<<c;
```

10. Ce se afișează în urma executării următoarei secvențe de instrucțiuni?


```
int a,b=2,c=1;
a=b=c, a=b==c;
cout<<a<<" "<<b<<" "<<c;
```

11. Ce se afișează în urma executării următoarei secvențe de instrucțiuni?

```
int a=1,b=2,c=1;
b=a--c, b=a=c;
cout<<a<<" "<<b<<" "<<c;
```

12. Ce se afișează în urma executării următoarei secvențe de instrucțiuni?

```
int a=1,b=2,c=3;
cout<<(a<b<c)<<" "<<(a<b>c);
```

13. Ce se afișează în urma executării următoarei secvențe de instrucțiuni?

```
unsigned char a=10,b=20,c=30;
a--a; cout<<(int)a<<endl;
a=b&c; cout<<(int)a<<endl;
a=b^c; cout<<(int)a<<endl;
a=b|c; cout<<(int)a<<endl;
```

14. Care dintre variabilele întregi a , b și c vor avea aceeași valoare după executarea următoarei instrucțiuni?

```
b=a+1, c=b-1, a=c+1;
```

15. Ce se afișează în urma executării următoarei instrucțiuni, dacă a și b sunt variabile întregi care au valoarea 5 și, respectiv, 7?

```
a-=b, b+=a, a=b-a;
```

16. Dacă c este o variabilă de memorie de tip **char** în care se memorează un caracter care este o literă mică, în urma executării instrucțiunii `cout<<c-( 'a'-'A' )`; se afișează litera mare corespunzătoare literei mici sau codul ASCII al literei mari?

Adevărat sau fals:

1. Operatorul `%` se poate aplica pe orice tip numeric.
2. Operatorul `/` nu se poate aplica pe tipul **char**.
3. Dacă variabila de memorie a este de tip **int**, rezultatul expresiei $3+a/2$ este de tip real.
4. Rezultatul expresiei $-5\%2$ este -1 .
5. Rezultatul expresiei $5\%-2$ este -1 .
6. Rezultatul expresiei $-5\%-2$ este -1 .
7. Expresia $-5\%-2$ produce eroare de compilare.
8. Dacă variabila de memorie a este de tip **float** și are valoarea 2, rezultatul expresiei $3+a/2$ este 4 și este de tip **int**.
9. Dacă variabila de memorie a este de tip **float** și are valoarea 2, iar variabila de memorie b este de tip **int** și are valoarea 3, rezultatul expresiei $b \geq a$ este 1 și este de tip **int**.
10. Dacă variabila de memorie a este de tip **char** și are valoarea 'a', iar variabila de memorie b este de tip **int** și are valoarea 3, rezultatul expresiei $a \geq b \ \&\& \ a$ este 1 și este de tip **int**.

11. Dacă variabila de memorie *a* este de tip **char** și are valoarea 'a', iar variabila de memorie *b* este de tip **int** și are valoarea 100, rezultatul expresiei $a \geq b \parallel a$ este 1 și este de tip **char**.
12. Dacă variabila de memorie *a* este de tip **float** și are valoarea 4, rezultatul expresiei $(a+2)/(a-2)$ este 3 și este de tip **int**.
13. Dacă variabila de memorie *a* este de tip **float** și are valoarea 2, rezultatul expresiei $(a+4)/3*a$ este 1 și este de tip **float**.
14. Dacă variabilele de memorie *a* și *b* sunt de tip **int** și au valoarea 2, respectiv 4, rezultatul expresiei $a/b+b/a$ este 2.5 și este de tip **float**.
15. Dacă variabilele de memorie *a* și *b* sunt de tip **int** și au valoarea 2, respectiv 4, rezultatul expresiei $++a/b--$ este 1 și este de tip **int**.
16. Dacă variabilele de memorie *a* și *b* sunt de tip **char**, expresia $a+b$ nu produce eroare de compilare, iar rezultatul este de tip **char**.
17. Dacă variabila de memorie *a* este de tip **char**, iar variabila de memorie *b* este de tip **int**, rezultatul evaluării expresiei $a=a+b$ este de tip **int**.

Alegeți:

1. Tipul rezultatului obținut în urma evaluării expresiei $15/3+10/3+5/3$ este:
a) **float** b) **double** c) **int** d) **unsigned**
2. Dacă *a*, *b*, *c* și *d* sunt trei variabile reale, prin ce instrucțiune i se atribuie variabilei *d* suma dintre media aritmetică a valorilor *a* și *b* și media aritmetică a valorilor *a* și *c*?
a) $d=a+b/2 + a+c/2$; c) $d=a+(b+c)/2$;
b) $d=(a+b+a+c)/4$; d) $d=(a+b+a+c)/3$;
3. Dacă *x*, *a* și *b* reprezintă variabile reale și $a < b$, ce expresie se utilizează într-un program pentru a verifica dacă valoarea variabilei $x \in [a, b]$?
a) $x > a \ \&\& \ x < b$ c) $x \geq a \ \&\& \ x \leq b$
b) $x \geq a \parallel x \leq b$ d) $a \leq x \leq b$
4. Dacă toate valorile variabilelor *a*, *b* și *x* sunt numere naturale, cum se poate atribui variabilei *x* restul împărțirii valorii variabilei *a* la valoarea variabilei *b*?
a) $x = a \% b$; c) $x \% = a/b$;
b) $(a \geq b)? x = a / b : x = b / a$; d) $(a \geq b)? x = a \% b : x = b \% a$;
5. Care dintre următoarele expresii este adevărată, dacă și numai dacă *x* este un număr natural impar, divizibil cu 5?
a) $x \geq 0 \parallel x \% 2 \neq 0 \parallel x \% 5 == 0$ c) $(x \geq 0) \parallel (x \% 2 \neq 1) \ \&\& \ (x \% 5 == 0)$
b) $(x \geq 0) \ \&\& \ (x \% 2 == 1 \parallel x \% 5 == 0)$ d) $x \geq 0 \ \&\& \ x \% 2 \neq 0 \ \&\& \ x \% 5 == 0$
6. Care dintre următoarele expresii este adevărată, dacă și numai dacă numărul întreg *x* este impar negativ?
a) $(x \% 2 == 1) \ \&\& \ (x < 0)$ c) $! ((x \% 2 == 0) \parallel (x \geq 0))$
b) $(x \% 2 \neq 0) \parallel (x < 0)$ d) $! ((x \% 2 == 0) \ \&\& \ (x \geq 0))$
7. Care dintre următoarele expresii se obține prin negarea expresiei $x \geq a \parallel x < b$:

- a) $x \leq a \parallel x > b$ c) $x \geq a \&\& x < b$
 b) $x < a \parallel x \geq b$ d) $x < a \&\& x \geq b$
8. Care dintre următoarele expresii se obține prin negarea expresiei $(!x \parallel a) \&\& (x \parallel !b)$:
 a) $(x \&\& !a) \&\& (!x \&\& b)$ c) $!(x \parallel a) \parallel !(x \parallel b)$
 b) $!(x \&\& a) \&\& !(x \&\& b)$ d) $(x \&\& !a) \parallel (!x \&\& b)$
9. Care dintre următoarele instrucțiuni nu mărește valoarea variabilei reale y cu jumătate din valoarea variabilei reale x ?
 a) $x/=2, y+=x;$ c) $y+=x=x/2;$
 b) $y+=x, x/=2;$ d) $y+=x/2;$
10. Care dintre următoarele expresii poate fi folosită pentru a verifica dacă două numere întregi a și b sunt amândouă pare?
 a) $(a-b)\%2==0$ c) $a*b\%4==0$
 b) $(a+b)\%2==0$ d) $(a\%2==0) \&\& (b\%2!=1)$

Rezolvați:

1. Scrieți un program care să afișeze codul ASCII al unui caracter introdus de la tastatură.
2. Scrieți un program care să calculeze aria și lungimea unui cerc. Dimensiunea razei se citește de la tastatură. Pentru numărul π se va folosi constanta simbolică PI.
3. Se citește un număr natural care reprezintă timpul exprimat în minute. Scrieți programul care afișează timpul exprimat în ore și minute.
4. Scrieți un program care să calculeze valoarea expresiei de la problema 8, pagina 31. Valoarea operandului se citește de la tastatură.
5. Scrieți un program care să calculeze valorile celor trei expresii de la problema 6, pagina 31. Valorile operanzilor se citesc de la tastatură.
6. Scrieți un program care să testeze un caracter introdus de la tastatură. Dacă este literă mare, să se afișeze mesajul "Literă mare", dacă este literă mică să se afișeze mesajul "Literă mică", altfel să se afișeze mesajul "Nu este literă".
7. Scrieți un program care afișează câte numere pare sunt într-un interval $[a, b]$. Valorile pentru a și b se citesc de la tastatură.
8. Scrieți un program care citește de la tastatură un număr natural cu trei cifre și care afișează apoi numărul obținut prin eliminarea cifrei din mijloc.
9. Scrieți un program care citește de la tastatură un număr natural cu patru cifre și care afișează pe câte un rând cifrele numărului, începând cu cifra cea mai semnificativă.
10. Scrieți un program care citește de la tastatură un număr format din două cifre și care afișează un mesaj prin care se precizează modul în care sunt ordonate cifrele: "Cifre ordonate crescător" sau "Cifre ordonate descrescător".
11. Scrieți un program care citește de la tastatură patru numere reale a, b, c și d cu $a < b$ și $c < d$ și care afișează intersecția intervalelor $[a, b]$ și $[c, d]$.

4.6.

Instru
al). În
vente
se rep
diție. P
se folo
în pse
contro
limbaju

1. Str

✓

2. Str

✓

✓

4.6.1.

Instruc
altfel...

Ps

da

sfă

Expresia
stantă lo

Instrucți

Pas 1. s

Pas 2. d

s

v

Exemplu

// Progr

// tast

#includ

void ma

4.6. Instrucțiunile de control

Instrucțiunile dintr-un program se execută în ordinea în care le-ați scris (secvențial). În marea majoritate a algoritmilor nu se execută toți pașii secvențial. Sunt frecvente cazurile în care se execută anumiți pași în funcție de o anumită condiție sau se repetă executarea anumitor pași atât timp cât este îndeplinită o anumită condiție. Pentru a putea implementa acest gen de algoritmi, în limbajul de programare se folosesc **instrucțiunile de control**. La fel ca și în cazul unui algoritm exprimat în pseudocod, ele specifică ordinea în care se execută instrucțiunile programului controlând **fluxul de execuție al programului**. Instrucțiunile de control definite în limbajul C++ sunt:

1. Structura alternativă:

✓ simplă

✓ generalizată

- if ... else

- switch ... case.

2. Structura repetitivă:

✓ condiționată anterior

✓ condiționată posterior

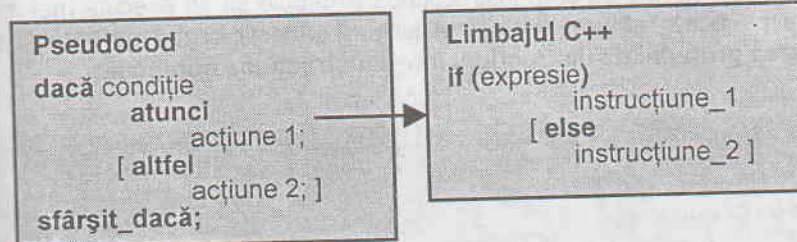
- while

- for

- do ... while

4.6.1. Instrucțiunea if ... else

Instrucțiunea **if ... else** implementează structura alternativă simplă (**dacă... atunci... altfel... sfârșit_dacă**). Sintaxa acestei instrucțiuni este:



Expresia trebuie să furnizeze un rezultat numeric care va fi interpretat ca o constantă logică: *True* sau *False*.

Instrucțiunea **if ... else** se execută astfel:

Pas 1. Se evaluează expresia **expresie**.

Pas 2. Dacă rezultatul expresiei este diferit de 0 (corespunde valorii logice *True*), se execută **instrucțiune_1**. Dacă rezultatul expresiei este 0 (corespunde valorii logice *False*), se execută **instrucțiune_2**.

Exemplul 1:

```

//Programul verifică dacă un număr întreg citit de la
// tastatură este un număr par.
#include <iostream.h>
void main()
```



```

(int x;
cout<<"x="; cin>>x;
if (n%2==0) //n%2==0 este expresia
    cout<<"Numărul este par "; // instrucțiune 1
else
    cout<<"Numărul nu este par "; // instrucțiune 2
}

```

Pentru **structura alternativă simplă cu o ramură vidă** nu se mai precizează **else** instrucțiune_2, instrucțiunea **if** având în acest caz sintaxa:
if (expresie) instrucțiune

Exemplul 2:

```

//Programul afișează modulul unui număr real citit de la
//tastatură.
#include <iostream.h>
void main()
{int x;
cout<<"x="; cin>>x;
if (x<0) // x<0 este expresia
    x=-x; // instrucțiune
cout<<"modulul = "<<x;
}

```

Observații:

1. Conform sintaxei instrucțiunii **if**, pe fiecare dintre ramurile structurii alternative nu se poate executa decât o singură instrucțiune. În cazul în care în algoritmul pentru rezolvarea problemei este nevoie să se execute mai mulți pași pe oricare dintre ramuri, înseamnă că și în instrucțiunea **if** trebuie să se execute mai multe instrucțiuni pe acea ramură, și nu una singură. Aceste cazuri se rezolvă prin încapsularea grupului de instrucțiuni într-o instrucțiune compusă.

Exemplul 3:

```

/*Programul afișează în ordine crescătoare două numere reale
a și b citite de la tastatură, folosind algoritmul de
interschimbare a două variabile de memorie*/
#include <iostream.h>
void main()
{float a,b,x;
cout<<"a="; cin>>a;
cout<<"b="; cin>>b;
if (b<a) // instrucțiunea compusă
    {x=a; a=b; b=x;}
cout<<a<<" "<<b;
}

```

2. Deoarece în limbajul C++ nu este implementat tipul logic, orice valoare numerică poate fi interpretată ca o constantă logică (*True* sau *False*). În cazurile în care în expresie se testează dacă o valoare este diferită de 0 (orice număr diferit de zero înseamnă *True*), expresia poate fi scrisă **condensat**.

Exemplul 4:

```

//Programul verifică dacă un număr întreg citit de la
//tastatură este un număr impar.

```



```
#include <iostream.h>
void main()
{int n;
cout<<"n="; cin>>n;
if (n%2)
    cout<<"Numărul este impar ";
else
    cout<<"Numărul nu este impar ";
```

În același mod se poate scrie și expresia pentru testarea unui număr par. Dacă $n\%2$ trebuie să fie egal cu 0 pentru ca numărul să fie par, înseamnă că $!(n\%2)$ trebuie să fie egal cu 1, adică diferit de 0 și, în loc să scriem expresia $!(n\%2)!=0$, putem condensa expresia cu $!(n\%2)$.

3. Pentru a implementa unii algoritmi se pot folosi instrucțiuni **if** imbricate. Să considerăm următorul exemplu. Se introduc de la tastatură numerele întregi a și b și un caracter care reprezintă o operație aritmetică. Să se calculeze valoarea lui e definită ca rezultat al aplicării operatorului aritmetic pe numerele a și b .

Exemplul 5:

```
#include <iostream.h>
void main()
{int a,b;
char c;
float e;
cout<<"a="; cin>>a;
cout<<"b="; cin>>b;
cout<<"c="; cin>>c;
if (c=='+')
    {e=a+b;
    cout<<"E= "<<e;}
else
    if (c=='-')
        {e=a-b;
        cout<<"E= "<<e;}
    else
        if (c=='*')
            {e=a*b;
            cout<<"E= "<<e;}
        else
            if (c=='/')
                {e=(float)a/b;
                cout<<"E= "<<e; }
            else
                cout<<"Operator gresit";
```



Construcțiile în care folosiți instrucțiuni **if** imbricate pot crea însă probleme atunci când una dintre structurile alternative este cu o ramură vidă. Să considerăm următorul exemplu. Se citesc trei numere întregi n , a și b . Să se calculeze valoarea lui x , astfel: dacă n este diferit de 0, x are valoarea lui $|a|$; altfel, are valoarea lui b . Să presupunem că se scrie următorul program:

Exemplul 6:

```
#include <iostream.h>
void main()
{int a,b,n,x;
  cout<<"n="; cin>>n;
  cout<<"a="; cin>>a;
  cout<<"b="; cin>>b;
  x=a;
  if (n)
    if (a<0) x=-a;
    else
      x=b;
  cout<<"x=" <<x;}
```

Scrierea indentată a acestor instrucțiuni nu rezolvă problema de sintaxă a limbajului. Conform sintaxei limbajului, cuvântul cheie **else** cel mai interior este asociat întotdeauna cu cuvântul cheie **if** cel mai interior. Așa cum a fost scris acest program, instrucțiunea $x=b$; se va executa dacă a este mai mare sau egal cu 0, și nu dacă n este diferit de 0. Pentru a corecta această greșală, vom încapsula, într-o instrucțiune compusă, instrucțiunea care se execută atunci când n este 0, chiar dacă aparent nu este necesar, fiind vorba de o singură instrucțiune:

Exemplul 7:

```
#include <iostream.h>
void main()
{int a,b,n,x;
  cout<<"n="; cin>>n;
  cout<<"a="; cin>>a;
  cout<<"b="; cin>>b;
  x=a;
  if (n)
    {if (a<0) x=-a; } // instrucțiunea compusă
    else
      x=b;
  cout<<"x=" <<x;}
```

4.6.2. Instrucțiunea switch ... case

Instrucțiunea **switch ... case** implementează structura alternativă generalizată (caz că... cazul... altfel... sfârșit_în_caz_că). Sintaxa acestei instrucțiuni este:

Pseudocod

```
în caz că selector
cazul v1: acțiune 1;
cazul v2: acțiune 2;
.....
cazul vi: acțiune i;
.....
cazul vn: acțiune n;
altfel   acțiune n+1;
sfârșit_în_caz_că;
```

Limbajul C++

```
switch (expresie)
```

```
{
  case exp_1: instrucțiune_1 break;
  case exp_2: instrucțiune_2 break;
  .....
  case exp_j: instrucțiune_j break;
  .....
  case exp_n: instrucțiune_n break;
  [default:  instrucțiune_n+1]
}
```

Expresia **expresie** trebuie să furnizeze un rezultat numeric întreg, iar expresiile care se evaluează pentru fiecare caz (*exp_i*) trebuie să fie constante întregi sau expresii constante cu valoare întreagă. Caracterul : este un separator între eticheta cazului (**case exp_i**) și instrucțiunea care se va executa (*instrucțiune_i*). Colecția de instrucțiuni etichetate este încapsulată într-un bloc delimitat de separatorii { }.

Instrucțiunea **switch ... case** se execută astfel:

- Pas 1.** Se evaluează expresia **expresie**.
 - Pas 2.** Dacă rezultatul obținut are valoarea **exp₁** se execută **instrucțiune₁** după care se trece la execuția instrucțiunii care urmează instrucțiunii **switch**, altfel se trece la pasul următor.
 - Pas 3.** Dacă rezultatul obținut are valoarea **exp₂** se execută **instrucțiune₂** după care se trece la execuția instrucțiunii care urmează instrucțiunii **switch**, altfel se trece la pasul următor.
-
- Pas n+1.** Dacă rezultatul obținut are valoarea **exp_n** se execută **instrucțiune_n** după care se trece la execuția instrucțiunii care urmează instrucțiunii **switch**, altfel se execută instrucțiunea care urmează după eticheta **default**, dacă aceasta există, după care se trece la execuția instrucțiunii care urmează instrucțiunii **switch**.

Exemplul 1:

```
//Programul de la Exemplul 5 al instrucțiunii if.
#include <iostream.h>
void main()
{int a,b;
 char c;
 float e;
 cout<<"a="; cin>>a;
 cout<<"b="; cin>>b;
 cout<<"c="; cin>>c;
 switch (c)
 {
 // începe blocul instrucțiunii switch
 case '+': {e=a+b; cout<<"E= "<<e;} break;
 case '-': {e=a-b; cout<<"E= "<<e;} break;
 case '*': {e=a*b; cout<<"E= "<<e;} break;
 case '/': {e=(float)a/b; cout<<"E= "<<e;} break;
 default: cout<<"Operator gresit";
 }
 // se închide blocul instrucțiunii switch
 }
```

Observații:

1. Toate expresiile corespunzătoare cazurilor (*exp_i*) trebuie să fie diferite între ele.
2. Eticheta **default** este opțională. Instrucțiunea atașată acestei etichete se execută numai dacă nu a fost îndeplinit nici un caz anterior.



Instrucțiunea **break**; este obligatorie. Semnificația ei este de a se întrerupe execuția instrucțiunii **switch ... case** și de a se trece la executarea următoarei instrucțiuni din program.

Dacă nu scrieți această instrucțiune, se continuă execuția instrucțiunii **switch ... case**. Deoarece cuvintele cheie **case** nu sunt decât niște etichete folosite pentru evidențierea cazurilor, continuarea execuției instrucțiunii **switch ... case** înseamnă de fapt executarea tuturor instrucțiunilor până la întâlnirea separatorului **}** care închide blocul instrucțiunii **switch ... case**.

Exemplul 2:

```
#include <iostream.h>
void main()
{int a,b;
 int n;
 cout<<"n (1,2 sau 3) ="; cin>>n;
 switch (n)
 {
 case 3: cout<<3<<" ";
 case 2: cout<<2<<" ";
 case 1: cout<<1<<" "; break;
 default: cout<<"Ati ales alt numar";
 }
 }
```

La execuția acestui program, în funcție de valoarea citită pentru *n*, se afișează:

- ✓ Dacă *n* are valoarea 3, se execută instrucțiunea cu eticheta **case 3**, după care se continuă cu execuția instrucțiunii cu eticheta **case 2**, și apoi cu a instrucțiunii cu eticheta **case 1**, după care se execută instrucțiunea **break** care întrerupe execuția instrucțiunii **switch ... case**. Programul va afișa **3 2 1**.
- ✓ Dacă *n* are valoarea 2, se execută instrucțiunea cu eticheta **case 2**, după care se continuă cu execuția instrucțiunii cu eticheta **case 1**, și apoi cu execuția instrucțiunii **break**, care întrerupe execuția instrucțiunii **switch ... case**. Programul va afișa **2 1**.
- ✓ Dacă *n* are valoarea 1, se execută instrucțiunea cu eticheta **case 1**, apoi instrucțiunea **break**, care întrerupe execuția instrucțiunii **switch ... case**. Programul va afișa **1**.
- ✓ Dacă *n* are o valoare diferită de 1, 2 sau 3, se execută instrucțiunea cu eticheta **default**. Programul va afișa **Ati ales alt numar**.

Observație. Instrucțiunea **switch ... case** poate fi folosită în locul unor instrucțiuni **if ... else** imbricate. Să considerăm următorul exemplu:

Exemplul 3:

```
/*Programul citește două numere întregi a și b și un caracter
c. Dacă pentru c se citește 1 sau 3, se afișează suma celor
două numere. Dacă pentru c se citește 2 sau 4, se afișează
diferența dintre valoarea mai mare și valoarea mai mică a
celor două numere. În restul cazurilor, se afișează mesajul
Alta optiune.*/
#include <iostream.h>
void main()
{int a,b;
 char c;
 cout<<"a= "; cin>>a;
 cout<<"b= "; cin>>b;
```

```
cout<<
if (c=
c
else
i
```

Programul

Exemplul

```
#include
void mai
{int a,b;
 char c;
 cout<<"
 cout<<"
 cout<<"
 switch
 {
 case
 case
 default
 }
 }
```

În funcție de

- ✓ Dacă *c* a
o instruc
cuția ins
și apoi
switch
eticheta
- ✓ Dacă *c* a
o instruc
cuția ins
loarea m
else), și
ționii **swi**
eticheta **c**
- ✓ Dacă *c* a
eticheta **d**

4.6.3. Inst

Instrucțiunea
pași, condițio
instrucțiuni es

```

cout<<"optiune (1,2,3 sau 4)= "; cin>>c;
if (c=='1' || c=='3')
    cout<<a+b;
else
    if (c=='2' || c=='4')
        if (a>b) cout<<a-b;
        else cout<<b-a;
    else
        cout<<"Alta optiune";}

```

Programul poate fi implementat și cu o structură **switch ... case**, astfel:

Exemplul 4:

```

#include <iostream.h>
void main()
{int a,b;
char c;
cout<<"a= "; cin>>a;
cout<<"b= "; cin>>b;
cout<<"optiune (1,2,3 sau 4)= "; cin>>c;
switch (c)
{
    case '1': case '3': cout<<a+b; break;
    case '2': case '4': if (a>b) cout<<a-b;
                        else cout<<b-a; break;
    default: cout<<"Alta optiune";
}
}

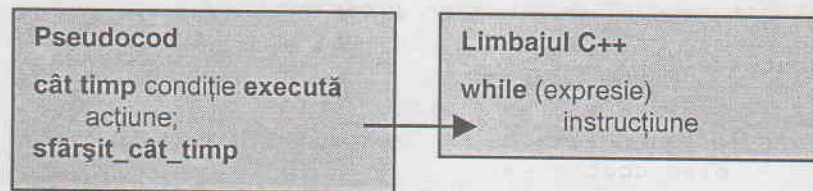
```

În funcție de valoarea citită pentru *c*, instrucțiunea **switch ... case** se execută astfel:

- ✓ Dacă *c* are valoarea '1', se execută instrucțiunea cu eticheta **case '1'**, care este o instrucțiune inexistentă (nu se execută nimic), după care se continuă cu execuția instrucțiunii cu eticheta **case '3'**, prin care se afișează suma dintre *a* și *b*, și apoi se execută instrucțiunea **break**, care întrerupe execuția instrucțiunii **switch ... case**. Dacă *c* are valoarea '3', se execută direct instrucțiunea cu eticheta **case '3'**, obținându-se același rezultat ca și în cazul valorii '1' pentru *c*.
- ✓ Dacă *c* are valoarea '2', se execută instrucțiunea cu eticheta **case '2'**, care este o instrucțiune inexistentă (nu se execută nimic), după care se continuă cu execuția instrucțiunii cu eticheta **case '4'**, prin care se afișează diferența dintre valoarea mai mare și valoarea mai mică a numerelor *a* și *b* (instrucțiunea **if ... else**), și apoi se execută instrucțiunea **break**, care întrerupe execuția instrucțiunii **switch ... case**. Dacă *c* are valoarea '4', se execută direct instrucțiunea cu eticheta **case '4'**, obținându-se același rezultat ca și în cazul valorii '2' pentru *c*.
- ✓ Dacă *c* are o valoare diferită de '1', '2', '3' sau '4', se execută instrucțiunea cu eticheta **default** și se va afișa **Alta optiune**.

4.6.3. Instrucțiunea while

Instrucțiunea **while** implementează structura repetitivă cu număr necunoscut de pași, condiționată anterior (**cât timp... execută... sfârșit_cât_timp**). Sintaxa acestei instrucțiuni este:



Instrucțiunea **while** se execută astfel:

- Pas 1.** Se evaluează expresia **expresie**.
Pas 2. Dacă rezultatul expresiei este diferit de 0 (corespunde valorii logice *True*) se execută **instrucțiune**, apoi se revine la **Pasul 1**; altfel, se trece la execuția instrucțiunii care urmează instrucțiunii **while**.

Ca și în cazul instrucțiunilor precedente, dacă trebuie să se repete executarea unui grup de instrucțiuni, în locul unei singure instrucțiuni, grupul de instrucțiuni va fi încapsulat într-o instrucțiune compusă.

Exemplul 1:

```
/*Se citesc mai multe numere întregi de la tastatură, până la
citirea numărului 0. Să se calculeze suma numerelor pare
citite.*/
#include <iostream.h>
void main()
{int a,s=0;
// variabila a se folosește pentru citirea numerelor
// variabila s se folosește pentru calcularea sumei și
// se inițializează cu 0
cout<<"a="; cin>>a;
while (a) // expresia a condensează expresia a!=0
{ // începutul instrucțiunii compuse
if (!(a%2)) s+=a;
// expresia !(a%2) condensează expresia a%2==0
// expresia s+=a condensează expresia s=s+a
cout<<"a="; cin>>a;
// se citește următorul număr în variabila a
} // sfârșitul instrucțiunii compuse
cout<<"suma= "<<s;
}
```

Exemplul 2:

```
/*Se citesc cifrele unui număr începând cu cifra cea mai
semnificativă. Să se afișeze numărul.*/
#include <iostream.h>
void main()
{unsigned long n=0;
unsigned int c;
// variabila c se folosește pentru citirea cifrelor
// variabila n se folosește pentru formarea numărului
cout<<"cifra= "; cin>>c;
while (c>=0 && c<=9) _
{n=n*10+c;
cout<<"cifra= "; cin>>c;}
```

```
cout<<"numarul= "<<n;
}
```

Exemplul 3:

```
/*Se citește un număr natural. Să se afișeze suma cifrelor.*/
#include <iostream.h>
void main()
{unsigned long n=0;
 int s=0;
 // variabila n se folosește pentru citirea numărului
 // variabila s se folosește pentru calcularea sumei cifrelor
 cout<<"numarul= "; cin>>n;
 while (n) _
 {s+=n%10; n/=10;}
 cout<<"suma= "<<n;
}
```

Observație:

Expresia care se evaluează poate să fie formată și dintr-o expresie compusă din mai multe expresii legate cu operatorul virgulă, iar instrucțiunea care se execută poate fi și instrucțiunea vidă. Programul din exemplul anterior poate fi rescris astfel:

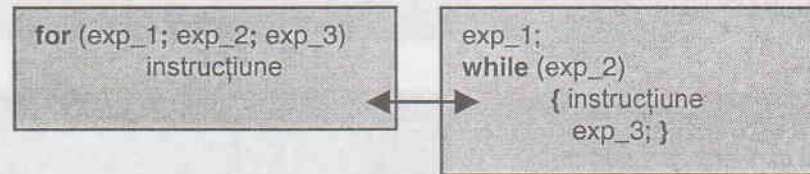
Exemplul 3_a:

```
#include <iostream.h>
void main()
{unsigned long n=0;
 int s=0;
 cout<<"numarul= "; cin>>n;
 while (s+=n%10,n/=10) // expresie compusă
 ; // se execută instrucțiunea vidă
 cout<<"suma= "<<n;
}
```

Instrucțiunea care se execută repetat în instrucțiunea **while** este instrucțiunea vidă (instrucțiunea **;**). Expresia compusă care se evaluează $s+=n\%10, n/=10$ este formată din două expresii legate de operatorul virgulă. Conform definiției limbajului, mai întâi se evaluează prima expresie $s+=n\%10$ (prin care se actualizează suma cifrelor) și apoi a doua expresie $n/=10$ (prin care se elimină ultima cifră din număr). Rezultatul expresiei compuse este dat de rezultatul ultimei expresii. Ultima expresie conținând operatorul de atribuire, rezultatul ei va fi dat de valoarea lui n . Așadar, execuția instrucțiunii **while** (instrucțiunea vidă) se va repeta atât timp cât n este diferit de 0. Actualizările sumei și a numărului n se fac prin intermediul expresiei care se testează, și nu prin intermediul unei instrucțiuni care se execută repetat.

4.6.4. Instrucțiunea for

Instrucțiunea **for** implementează tot o structură repetitivă cu număr necunoscut de pași, condiționată anterior, la fel ca și instrucțiunea **while**. Avantajul ei este că permite o scriere mai condensată decât instrucțiunea **while**. Sintaxa acestei instrucțiuni este:



Cele trei expresii se folosesc pentru implementarea **procesului de control** al executării repetate a instrucțiunii și corespund celor trei acțiuni ale acestui proces:

- ✓ **exp_1**, pentru **inițializare**, prin care se stabilește starea dinaintea primei execuții a instrucțiunii,
- ✓ **exp_2**, pentru **testare**, prin compararea stării curente cu starea care termină procesul de repetare și are rolul de a termina executarea repetată a instrucțiunii; trebuie să fie o expresie numerică, al cărei rezultat să poată fi interpretat ca o constantă logică (*True* sau *False*),
- ✓ **exp_3**, pentru **modificare**, prin schimbarea stării curente, astfel încât să se avanseze către starea finală, care încheie procesul de repetare a executării instrucțiunii.

Instrucțiunea **for** se execută astfel:

Pas 1. Se evaluează expresia **exp_1**.

Pas 2. Se evaluează expresia **exp_2**. Dacă rezultatul expresiei este diferit de 0 (corespunde valorii logice *True*) se execută **instrucțiune**, altfel se trece la execuția instrucțiunii care urmează instrucțiunii **for**.

Pas 3. Se evaluează expresia **exp_3** și se revine la **Pas 2**.

Exemplul 1:

```
/*Programul este echivalent cu programul de la Exemplul 3 al
  instrucțiunii while*/
#include <iostream.h>
void main()
{unsigned longint n;
 int s;
 cout<<"n="; cin>>n;
 for (s=0; n; n/=10)
     s+=n%10;      // instrucțiunea
 cout<<"suma= "<<s;
 }
```

Expresia pentru inițializare **exp_1** este $s=0$, expresia pentru testare **exp_2** este n , expresia pentru modificare **exp_3** este $n/=10$. Instrucțiunea **for** se execută astfel:

Pas 1. Se evaluează expresia **exp_1**, adică se inițializează suma cifrelor cu 0.

Pas 2. Se evaluează expresia **exp_2**. Dacă n este diferit de 0 (nu s-au eliminat toate cifrele din număr) se execută instrucțiunea $s+=n\%10$; prin care se actualizează valoarea lui s (prin adunarea ultimei cifre la sumă). Dacă n este 0 (s-au eliminat toate cifrele din număr) se termină execuția instrucțiunii **for** și se trece la execuția instrucțiunii `cout<<"suma= "<<s;`.

Pas 3. Se evaluează expresia **exp_3** prin care se modifică valoarea lui n (prin eliminarea ultimei cifre din număr). Se revine la **Pas 2**.

Instrucțiunea **for** din exemplul precedent mai poate fi scrisă în următoarele variante:

Exemplul 1_a:

```
for (s=0; n; s+=n%10, n/=10)
    // se execută instrucțiunea vidă
```

Expresia pentru inițializare **exp_1** este $s=0$, expresia pentru testare **exp_2** este n , iar expresia pentru modificare **exp_3** este expresia compusă $s+=n\%10, n/=10$. La execuție apar următoarele diferențe:

Pas 3. Se evaluează expresia **exp_3** prin care se actualizează mai întâi valoarea lui s (prin adunarea ultimei cifre la sumă), iar apoi se modifică valoarea lui n (prin eliminarea ultimei cifre din număr). În **exp_3** au fost condensate o instrucțiune care face parte din corpul structurii repetitive și acțiunea de modificare a procesului de control. Se revine la **Pas 2**.

Exemplul 1_b:

```
for (s=n%10; n/=10; s+=n%10);
```

Expresia pentru inițializare **exp_1** este $s=n\%10$, expresia pentru testare **exp_2** este $n\neq 0$, iar expresia pentru modificare **exp_3** este expresia $s+=n\%10$. La execuție apar următoarele diferențe:

Pas 1. Se evaluează expresia **exp_1**, adică se inițializează suma cifrelor cu prima cifră a numărului.

Pas 2. Se evaluează expresia **exp_2**, adică rezultatul operatorului de atribuire care este dat de valoarea lui n obținută prin eliminarea ultimei cifre. Dacă n este diferit de 0 (nu s-au eliminat toate cifrele din număr) se execută instrucțiunea vidă. Dacă n este 0 (s-au eliminat toate cifrele din număr) se termină execuția instrucțiunii **for**. În **exp_2** au fost condensate acțiunile de testare și de modificare ale procesului de control.

Pas 3. Se evaluează expresia **exp_3** prin care se modifică mai întâi valoarea lui s (prin adunarea ultimei cifre la sumă). În **exp_3** este instrucțiunea care face parte din corpul structurii repetitive. Se revine la **Pas 2**.

Ținând cont de modul în care se execută instrucțiunea **for**, în Exemplul 1_b suma s -a inițializat cu prima cifră, și nu cu 0 ca în Exemplul 1 și Exemplul 1_a, deoarece în aceste exemple mai întâi se adună cifra la sumă și apoi se elimină din număr, iar în Exemplul 1_b mai întâi se elimină o cifră din număr și apoi se adună o cifră la număr.

Observație:

Din punct de vedere sintactic, cele trei componente ale instrucțiunii **for** sunt expresii. Oricare dintre ele și chiar toate trei pot fi expresii vide. Caracterele **;** sunt însă obligatorii. De exemplu, instrucțiunea **for (;;);** este corectă din punct de vedere sintactic. Ea execută un **ciclu infinit** (un ciclu care nu se termină). Programul care conține un ciclu infinit se numește **program care ciclează la infinit**.



Dacă veți omite din instrucțiunea **for** expresia **exp_2**, instrucțiunea va executa un **ciclu infinit**.

În următoarele două exemple, **exp_3** este expresia vidă. Acțiunea de modificare a procesului de control se execută în corpul structurii repetitive prin citirea unei valori pentru variabila **a**, respectiv pentru variabila **c**.

Exemplul 2:

```
/*Programul echivalent cu programul de la Exemplul 1 al
instrucțiunii while*/
#include <iostream.h>
void main()
{int a,s;
cout<<"a="; cin>>a;
for (s=0;a;)
{
// începutul instrucțiunii compuse
if (!(a%2)) s+=a;
cout<<"a="; cin>>a;
}
// sfârșitul instrucțiunii compuse
cout<<"suma= "<<s; }
```

Exemplul 3:

```
/*Programul este echivalent cu programul de la Exemplul 2 al
instrucțiunii while*/
#include <iostream.h>
void main()
{unsigned long n;
unsigned int c;
cout<<"cifra= "; cin>>c;
for (n=0; c>=0 && c<=9;)
{n=n*10+c; cin>>c;}
cout<<"numarul= "<<n; }
```

Observație:

Crearea unui flux de date este considerată o expresie. În exemplul următor, expresia compusă folosită pentru **exp_3** conține o expresie pentru crearea fluxului **cin>>**:

Exemplul 3_a

```
/*Programul este echivalent cu programul de la Exemplul 3 al
instrucțiunii for*/
#include <iostream.h>
void main()
{unsigned long n;
unsigned int c;
for (c=0,n=0; c>=0 && c<=9; n=n*10+c, cin>>c);
cout<<"numarul= "<<n; }
```

Expresia pentru inițializare **exp_1** este **c=0,n=0**, expresia pentru testare **exp_2** este **c>=0 && c<=9**, iar expresia pentru modificare **exp_3** este expresia compusă **n=n*10+c, cin>>c**. Instrucțiunea **for** se execută astfel:

Pas 1. Se evaluează expresia **exp_1**, adică se inițializează cifra și numărul cu 0.

Pas 2. Se evaluează expresia **exp_2**. Dacă *c* este o cifră (un număr cu valoare între 0 și 9), se execută instrucțiunea vidă. Dacă *n* nu este o cifră, se termină execuția instrucțiunii **for**.

Pas 3. Se evaluează expresia **exp_3**, prin care se actualizează mai întâi valoarea numărului *n*, iar apoi se modifică valoarea cifrei *c* (prin citirea unui nou număr). În **exp_3** au fost condensate instrucțiunea care face parte din corpul structurii repetitive și acțiunea de modificare a procesului de control. Se revine la **Pas 2**.

Observație:

În expresia **exp_1** puteți să declarați variabile de memorie și să le inițializați:

Exemplul 3_b

```
/*Programul este echivalent cu programul de la Exemplul 3 al
instrucțiunii for*/
#include <iostream.h>
void main()
{for (unsigned long c=0,n=0; c>=0 && c<=9; n=n*10+c,cin>>c);
  cout<<"numarul= "<<n;
}
```

Nu puteți să declarați variabile de tipuri diferite. Exemplul următor este greșit din punct de vedere sintactic și va produce eroare la compilare:

```
for(unsigned long n=0,unsigned c=0;c>=0&& c<=9;n=n*10+c,cin>>c);
```

Recomandare:

Versiunile *a* și *b* ale programelor anterioare nu sunt greșite din punct de vedere sintactic, și nici din punct de vedere logic. Avantajul instrucțiunii **for** față de instrucțiunea **while** este că **centralizează în cele trei expresii cele trei acțiuni ale procesului de control**. Acest avantaj este evident mai ales în cazul **instrucțiunilor repetitive imbricate**. Din această cauză este bine să vă obișnuiți să folosiți cele trei expresii numai pentru cele trei acțiuni ale procesului de control.

Observație:

În limbajul C++ nu este definită o instrucțiune specială pentru **structura repetitivă cu număr cunoscut de pași**. Instrucțiunea cea mai adecvată pe care o puteți folosi pentru implementarea acestei structuri dintr-un algoritm este instrucțiunea **for**.

Pseudocod

```
pentru contor = vi, vf [pas v] execută
    acțiune;
sfârșit_pentru;
```

Limbajul C++

```
for (contor=vi;contor<=vf; contor+=v)
    instrucțiune
```

Expresia pentru inițializare **exp_1** se folosește pentru inițializarea contorului, expresia pentru testare **exp_2** se folosește pentru a compara valoarea contorului cu valoarea finală *vf*, iar expresia **exp_3** se folosește pentru modificarea contorului prin incrementare sau decrementare.

Exemplul 4:

```

/*Se citește un număr natural n și un șir de n numere
  întregi. Să se afișeze suma numerelor pare citite în șir.*/
#include <iostream.h>
void main()
{int n,i,a,s;
  // variabila i se folosește pentru contor
  // variabila n reprezintă valoarea finală a contorului
  // variabila a se folosește pentru citirea numerelor
  // variabila s se folosește pentru calcularea sumei
  // inițializarea variabilelor i și s se face în expresia
  // exp 1 a instrucțiunii for
  cout<<"n="; cin>>n;
  for (i=1,s=0; i<=n; i++)
    {cout<<"a="; cin>>a;
     if (!(a%2)) s+=a; }
  cout<<"suma= ";<<s;
}

```

Observație: Numărarea executării repetate a instrucțiunii **for** se poate face atât prin incrementarea cât și prin decrementarea contorului, de la o valoare inițială până la o valoare finală. În exemplul precedent, numărarea s-a făcut prin incrementarea contorului de la valoarea inițială 1 până la valoarea finală n . În exemplul următor, numărarea se face prin decrementarea contorului de la valoarea inițială n până la valoarea finală 1:

Exemplul 4 a:

```

//Programul este echivalent cu programul de la Exemplul 4
#include <iostream.h>
void main()
{int n,i,a,s;
  cout<<"n="; cin>>n;
  for (i=n,s=0; i>=1; i--)
    {cout<<"a="; cin>>a;
     if (!(a%2)) s+=a; }
  cout<<"suma= ";<<s;
}

```

Alegerea modului de numărare se face în funcție de cât de adecvat este acest mod pentru implementarea algoritmului care rezolvă problema. Pentru exemplul prezentat este evident că mai adecvată este numărarea prin incrementare. În următorul exemplu, mai adecvată este numărarea prin decrementare:

Exemplul 5:

```

/*Se citește un număr natural n. Să se afișeze în ordine inver-
  să toate numerele naturale pare mai mici sau egale cu n.*/
#include <iostream.h>
void main()
{int n,i;
  cout<<"n="; cin>>n;
  if (n%2) n--; // dacă n este impar se decrementează
  for (i=n;i>=2;i--,i--) // sau for (i=n;i>=2;i-=2)
    cout<<i<<" ";
}

```

În următorul exemplu se folosesc două instrucțiuni **for** imbricate:

Exemplul 6:

```
/*Se citește un număr natural n. Să se afișeze toate numerele
perfecte mai mici decât n*/
#include <iostream.h>
#include <math.h>
void main()
{int n,i,j,s;
  /* variabila i este contorul primului for și se folosește
  pentru a genera numerele care se verifică dacă sunt
  numere perfecte (numerele de la 2 până la n) */
  /* variabila j este contorul celui de al doilea for și se
  folosește pentru a genera divizorii posibili ai
  numărului i (numerele de la 2 până la sqrt(i) */
  // variabila n reprezintă valoarea finală a contorului i
  /* variabila s se folosește pentru calcularea sumei
  divizorilor numărului j și s-a inițializat cu 1 deoarece
  toate numerele testate au acest divizor */
  cout<<"n="; cin>>n;
  for (i=2;i<=n;i++)
    {for (j=2,s=1;j<=sqrt(i);j++)
      if(i%j==0) s+=j+i/j;
      if (i==s) cout<<i<<" "; }
}
```

În primul **for** se execută două instrucțiuni: o instrucțiune **for**, prin care se calculează suma divizorilor numărului i , și o instrucțiune **if**, prin care se testează dacă suma divizorilor este egală cu numărul i . Cele două instrucțiuni au fost încapsulate într-o instrucțiune compusă. În al doilea **for** nu se execută decât o instrucțiune **if** care verifică dacă un număr j este divizor al lui i . Ambele instrucțiuni **if** au o ramură vidă. În fiecare instrucțiune **if** nu se execută decât o singură instrucțiune.

Pentru a vedea avantajele folosirii instrucțiunii **for** în locul instrucțiunii **while** pentru structurile repetitive cu număr cunoscut de pași, același algoritm a fost implementat și cu instrucțiunea **while**.

Exemplul 6_a:

```
#include <iostream.h>
#include <math.h>
void main()
{int n,i=2,s,j;
  cout<<"n="; cin>>n;
  while (i<=n)
  {
    j=2; s=1;
    //Inițializările se fac înaintea instrucțiunii while
    while (j<=sqrt(i))
    {
      if(i%j==0) s+=j+i/j;
      j++;
      // Incrementarea contorului se face în corpul ciclului
    }
    if (i==s) cout<<i<<" ";
    i++;
  }
}
```



```

    i++;
  }
}

```

Observație. Variabila contor a instrucțiunii **for** poate avea orice tip numeric (**int**, **char**, **float** sau **double**):

Exemplul 7:

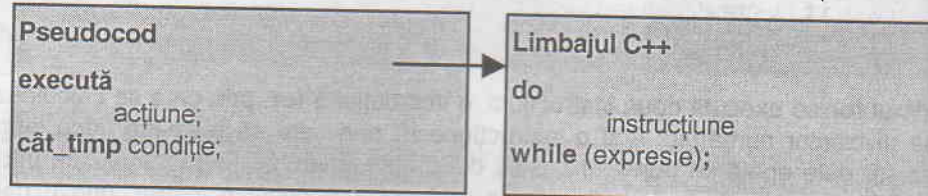
```

#include <iostream.h>
void main()
{for (char c='a';c<='z';c++) cout<<c<<" ";
  //Afișează în ordine alfabetică literele mici.
  cout<<endl;
  for (float x=0.9;x>0;x-=0.1) cout<<x<<" ";
  //Afișează descrescător numerele reale cu o cifră zecimală.
}

```

4.6.5. Instrucțiunea **do ... while**

Instrucțiunea **do ... while** implementează structura repetitivă cu număr necunoscut de pași condiționată posterior (**execută... cât timp**). Sintaxa acestei instrucțiuni este:

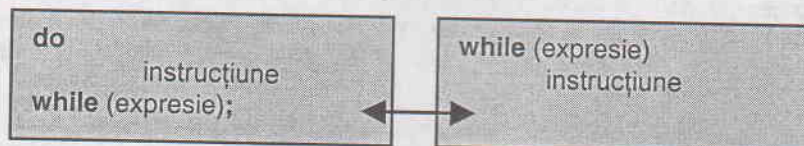


Instrucțiunea **do... while** se execută astfel:

Pas 1. Se execută **instrucțiune**.

Pas 2. Se evaluează **expresie**. Dacă rezultatul expresiei este diferit de 0 (corespunde valorii logice **True**), se revine la **Pasul 1**; altfel, se trece la execuția instrucțiunii care urmează instrucțiunii **do... while**.

Structurile repetitive cu număr necunoscut de pași pot fi implementate atât cu instrucțiunea **while**, cât și cu instrucțiunea **do... while**:



Exemplul 1:

```

/*Programul este echivalent cu programul de la Exemplul 3 al
  instrucțiunii while*/
#include <iostream.h>
void main()
{unsigned longint n;
  int s=0;
  cout<<"numarul= "; cin>>n;
}

```



```

do
{
    s+=n%10; n/=10; // instrucțiunea compusă
}
while (n); // expresia n condensează n!=0
cout<<"suma= "<<s;
}

```

Observație: Spre deosebire de instrucțiunea **while**, la instrucțiunea **do ... while**, **instrucțiunea** se execută cel puțin o dată. Din această cauză, uneori puteți să aveți probleme la implementarea unui algoritm. Să considerăm următorul exemplu:

Exemplul 2:

```

/*Cerințele algoritmului implementat cu acest program sunt
cele de la Exemplul 2 al instrucțiunii while*/
#include <iostream.h>
void main()
{unsigned long n=0;
 unsigned int c;
 cout<<"cifra= "; cin>>c;
 do
 {n=n*10+c;
  cout<<"cifra= "; cin>>c;}
 while (c>=0 && c<=9);
 cout<<"numarul= "<<n;
}

```

Pentru ca acest program să fie echivalent cu cel care a implementat același algoritm cu instrucțiunea **while**, trebuie să producă aceleași rezultate pentru același set de date de intrare. Să considerăm însă următorul caz: prima valoare care se citește pentru variabila *c* nu este o cifră (de exemplu, 11). În cazul implementării cu instrucțiunea **while**, testarea făcându-se înainte de executarea instrucțiunii compuse, instrucțiunea compusă nu se execută, și valoarea variabilei *n* este 0. În cazul instrucțiunii **do...** **while** mai întâi se execută instrucțiunea compusă și după aceea se testează noua valoare citită pentru variabila *c* (care să presupunem că nu este o cifră). Valoarea afișată pentru *n* va fi 11. Corectarea acestui program se poate face în două moduri: fie executarea instrucțiunii **do ... while** nu afectează valoarea variabilei *n* atunci când prima valoare citită pentru *c* nu este o cifră (Exemplul 2_a), fie nu se execută instrucțiunea **do ... while** dacă prima valoare citită pentru *c* nu este o cifră (Exemplul 2_b):

Exemplul 2_a

```

#include <iostream.h>
void main()
{unsigned long n=0;
 unsigned int c=0;
 do
 {n=n*10+c;
  cout<<"cifra= "; cin>>c;}
 while (c>=0 && c<=9);
 cout<<"numarul= "<<n;
}

```

Exemplul 2_b

```

#include <iostream.h>
void main()
{unsigned long n=0;
 unsigned int c;
 cout<<"cifra= "; cin>>c;
 if (c>=0 && c<=9)
 do
 {n=n*10+c;
  cout<<"cifra= "; cin>>c;}
 while (c>=0 && c<=9);
 cout<<"numarul= "<<n;
}

```




Greșeli pe care le puteți face atunci când scrieți în program instrucțiunile de control și care vor duce la apariția **erorilor de sintaxă** sau a avertismentelor la compilare:

1. Ați uitat să scrieți caracterul separator ; la sfârșitul instrucțiunii **do... while**.
2. Ați uitat să scrieți caracterul ; care separă cele trei expresii ale instrucțiunii **for**.
3. Ați uitat să delimitați între paranteze rotunde () expresia care se testează într-o instrucțiune de control.
4. Nu ați închis toate instrucțiunile compuse (parantezele { } nu sunt perechi).
5. Ați folosit operatorul de atribuire (=) în locul operatorului relațional egal (==).



Greșeli pe care le puteți face atunci când scrieți în program instrucțiunile de control și care vor duce la un **program care ciclează la infinit**:

1. Ați uitat să inițializați o variabilă a cărei valoare este folosită pentru contorizarea ciclului. Dacă nu inițializați variabila, ea va avea ca valoare inițială o **valoare reziduală**, și, chiar dacă îi veți modifica valoarea în instrucțiune, este posibil ca ea să nu ajungă să aibă valoarea pentru terminarea ciclului.
2. În expresia prin care testați terminarea executării ciclului ați folosit operatorul de atribuire (=) în locul operatorului relațional egal (==). De exemplu, dacă scrieți **for (i=1; i=n; i++)** în loc de **for (i=1; i==n; i++)**, dacă *n* are o valoare diferită de 0 (cazul cel mai probabil), prima instrucțiune **for** va cicla la infinit.
3. Ați uitat să scrieți instrucțiunea pentru acțiunea de modificare a procesului de control (cazurile cele mai frecvente sunt cele de actualizare a contorului prin incrementare sau decrementare sau cele de citire a unei variabile de memorie de a cărei valoare depinde terminarea execuției instrucțiunii repetitive).
4. Nu ați închis corect instrucțiunile compuse (parantezele { } sunt perechi dar nu sunt scrise corect). Dacă nu închideți corect o instrucțiune compusă este posibil ca instrucțiunea corespunzătoare acțiunii de modificare din procesul de control să nu se mai execute în instrucțiunea repetitivă.
5. Nu ați scris în instrucțiunea **for** expresia **exp_2**.
6. Ați modificat într-o instrucțiune din corpul structurii repetitive cu număr cunoscut de pași, implementată cu instrucțiunea **for**, fie variabila folosită pentru contorizare, fie variabila folosită pentru valoarea finală (de exemplu, prin modificarea valorii finale, contorul nu mai reușește să aibă valoarea acesteia sau, în cazul în care terminarea execuției instrucțiunii **for** este decisă de egalitatea dintre variabila contor și o variabilă a cărei valoare nu se modifică, este posibil ca, prin modificarea variabilei contor în corpul structurii, să depășiți valoarea finală).
7. Ați scris incorect expresia prin care verificați terminarea unei structuri repetitive.



1. Ați car
2. Ați Ace
3. Ați stru
4. Nu sur
5. Ați ca
- ins
5. Ați in
- i==



Răsp

1. Cu liza
2. Ce cu
3. Ce no
4. Cu cu
5. Ce
6. Ur a a
- int
- cout
- if (
7. Ca pe
8. So
- int
- cout
- if (
9. Se sir



Greșeli pe care le puteți face atunci când scrieți în program instrucțiunile de control și care vor duce la un **program care nu furnizează rezultatele dorite**:

1. Ați uitat să inițializați o variabilă de a cărei valoare depinde rezultatul final și care se calculează printr-un proces iterativ.
2. Ați folosit operatorul de atribuire (=) în locul operatorului relațional egal (==). Această greșeală în expresia unei instrucțiuni **if** poate determina execuția instrucțiunii de pe cealaltă ramură și nu a celei care a fost gândită în algoritm.
3. Ați uitat să scrieți o instrucțiune **break** într-o instrucțiune **switch**.
4. Nu ați închis corect instrucțiunile compuse (parantezele { } sunt perechi dar nu sunt scrise corect). Dacă nu închideți corect o instrucțiune compusă, este posibil ca instrucțiunea prin care obțineți iterativ rezultatul final să nu se mai execute în instrucțiunea repetitivă.
5. Ați scris separatorul ; după expresia instrucțiunii **for** sau **while** în cazul în care în structura repetitivă nu se execută instrucțiunea vidă (de exemplu **for** (i=1; i==n; i++); sau **while** (n!=0);).

Evaluare

Răspundeți:

1. Cu ce instrucțiuni de control puteți implementa o structură alternativă generalizată? Care dintre aceste instrucțiuni este mai adecvată implementării?
2. Ce instrucțiuni de control puteți folosi pentru a implementa structura repetitivă cu număr necunoscut de pași?
3. Ce instrucțiune de control definește numai o structură repetitivă cu număr cunoscut de pași?
4. Cu ce instrucțiuni de control puteți implementa o structură repetitivă cu număr cunoscut de pași? Care dintre aceste instrucțiuni este mai adecvată implementării?
5. Ce cuvinte cheie se folosesc în instrucțiunile de control? Care este semnificația lor?
6. Următoarea secvență de instrucțiuni trebuie să afișeze un mesaj dacă variabila a are valoarea 10. Corectați greșeala.

```
int a;
cout<<"a= "; cin>>a;
if (a=10) cout<<"Mesaj";
```

7. Care este cea mai mare valoare pe care o poate avea variabila întreagă a pentru ca instrucțiunea **if** (a<8-3*a) **cout<<"Mesaj";** să afișeze mesajul?
8. Scrieți instrucțiunea **switch** echivalentă cu următoarea instrucțiune **if**.

```
int n;
cout<<"n= "; cin>>n;
if (n%2==0) cout<<"Numar par"; else cout<<"Numar impar";
```

9. Se dă următoarea secvență de instrucțiuni. Scrieți secvența echivalentă: a) folosind o singură instrucțiune **if**; b) folosind o instrucțiune **switch**.


```
int x,y;
cout<<"x= "; cin>>x;
if (x==0) y=1; else
    if (x>=3&& x<=5) y=1; else
        if (x==2||x==6) y=0; else
            if (x==1) y=1; else if (x==7) y=0;
```

10. Se dă următoarea secvență de instrucțiuni. Scrieți secvența echivalentă folosind două instrucțiuni if.

```
int a,b,c,x;
cin>>a>>b>>c;
if (a>=b) if (a>=b) x=1; else x=2;
    else if (b>=c) if (a>=c) x=1; else x=3;
    else if (a>=b) x=2; else x=3;
```

11. Următoarea secvență de instrucțiuni trebuie să afișeze maximul dintre a și b. Corectați greșeala.

```
int a,b;
cout<<"a= "; cin>>a; cout<<"b= "; cin>>b;
if (a>b) cout<<a else cout<<b;
```

12. Următoarea secvență de instrucțiuni trebuie să afișeze modulul unui număr real x. Corectați greșeala.

```
float x;
cout<<"x= "; cin>>x;
if x<0 x=-x; cout<<x;
```

13. Următoarea secvență de instrucțiuni trebuie să afișeze valoarea variabilei n (putând lua valorile 1, 2 sau 3). Corectați greșelile.

```
int n;
cout<<"n= "; cin>>n;
switch (n)
{
    case 3: cout<<"3";
    case 2: cout<<"2";
    case 1: cout<<"1";
}
```

14. Următoarea instrucțiune trebuie să afișeze primele 10 numere naturale. Corectați greșelile.

```
for (int i=0; i<10; ) cout<<i<<" ";
```

15. Următoarea instrucțiune trebuie să afișeze primele 10 numere naturale. Corectați greșeala.

```
for (int i=1; i++;) cout<<i<<" ";
```

16. Următoarea instrucțiune trebuie să afișeze primele 10 numere naturale. Corectați greșeala.

```
for (int i=1; i<=10; i++); cout<<i<<" ";
```

17. Următoarea secvență de instrucțiuni trebuie să afișeze primele 10 numere naturale. Corectați greșeala.

```
int i=1;
while (i<=10) cout<<i<<" ";
```

18. Următoarea instrucțiune trebuie să afișeze pe același rând de 10 ori cifra 5. Corectați greșeala.

```
for (int i=1; i<=10; i++) {i=5; cout<<i;}
```

19. Următoarea instrucțiune trebuie să afișeze pe același rând cifrele de la 0 la 9. Corectați greșelile.


```
for (int i=0; i<=10; i++); cout<<i;
```

20. Următoarea secvență de instrucțiuni trebuie să afișeze numerele impare mai mici decât 10. Corectați greșeala.

```
int i=1;
while (i=10) {cout<<i<<" "; i++; i++;}
```

21. Ce afișează următoarea secvență de instrucțiuni?

```
for (int i=5; i>0; i--); cout<<i;
```

22. Ce afișează următoarea secvență de instrucțiuni?

```
while (1); cout<<1;
```

23. Ce afișează următoarea secvență de instrucțiuni?

```
unsigned a;
int s=0;
while (s>=0) {cout<<"a= "; cin>>a; s+=a;} cout<<a;
```

24. Următoarea secvență de instrucțiuni trebuie să afișeze suma primelor n numere pare. Corectați greșelile.

```
for (int i=0, s=0; i<=n; i+=2) s+=i; cout<<s;
```

25. Următoarea secvență de instrucțiuni trebuie să afișeze suma numerelor pare mai mici decât n . Corectați greșeala.

```
int i=2, s=0; do {i++, i++; s+=i;} while (i<=n); cout<<s;
```

26. Ce se afișează în urma execuției următoarei secvențe de instrucțiuni? Scrieți o secvență de instrucțiuni echivalentă care să implementeze mai eficient algoritmul.

```
for (i=1; i<=5; i+=2) {j=5; do j--; while (j>=i); cout<<j;}
```

27. Ce se afișează în urma execuției următoarei secvențe de instrucțiuni? Scrieți o secvență de instrucțiuni echivalentă care să implementeze mai eficient algoritmul.

```
for (i=10, j=0; i;) {--i, --i; j=i++; cout<<j;}
```

28. Ce se afișează în urma execuției următoarei secvențe de instrucțiuni?

```
for (i=10; i>0; i--) {--i, --i; cout<<i;}
```

29. Ce se afișează în urma execuției următoarei secvențe de instrucțiuni?

```
for (n=10, i=1; i<=n; i++) --n, --n, n--; cout<<i;
```

30. Ce se afișează în urma execuției următoarei secvențe de instrucțiuni?

```
for (n=2; n<=9; n+=2); i=9; while (i<=n) n++, i++; cout<<i<<n;
```

Adevărat sau fals:

1. Selectorul instrucțiunii **switch** poate fi de tip **float**.
2. Orice instrucțiune **if** poate fi înlocuită cu o instrucțiune **switch**.
3. Instrucțiunea **switch** este un bloc de instrucțiuni **if** imbricate.
4. Următoarele instrucțiuni **if** și **switch** sunt echivalente pentru orice numere a și b :

```
if (a==b)
    cout<<"Egalitate";
else
    cout<<"Inegalitate";
```

```
switch (a){
    case b:
        cout<<"Egalitate"; break;
    default cout<<"Inegalitate";}
```

5. Instrucțiunea **for** definește o structură repetitivă cu număr cunoscut de pași.
6. În orice program, dacă se înlocuiește o instrucțiune **while** cu instrucțiunea **do ... while** echivalentă, se obține un program echivalent.

7. În orice program, dacă se înlocuiește o instrucțiune **for** cu instrucțiunea **while** echivalentă se obține un program echivalent.

Alegeți:

1. Următoarea secvență de instrucțiuni afișează:

```
unsigned a,b,c,x;
if (a>b) if (b>c) x=b; else x=c;
        else
            if (a>c) x=a; else x=c;
cout<<x;
```

- a) min (max (a,c), max(b,c)) c) max (min (a,c), min(b,c))
b) max (max (a,c), max(b,c)) d) min (min (a,c), min(b,c))

2. Instrucțiunea **while** implementează:

- a) o structură repetitivă cu număr cunoscut de pași;
b) o structură alternativă generalizată;
c) o structură repetitivă condiționată anterior;
d) o structură repetitivă condiționată posterior.

3. Instrucțiunea **for** implementează:

- a) o structură repetitivă cu număr cunoscut de pași;
b) o structură alternativă generalizată;
c) o structură repetitivă condiționată anterior;
d) o structură repetitivă condiționată posterior.

4. Pentru implementarea structurii repetitive cu număr cunoscut de pași nu puteți folosi instrucțiunea:

- a) **for** b) **while** c) **do...while** d) **switch**

5. Care trebuie să fie valoarea inițială a unei variabile întregi *i* pentru ca instrucțiunea **while (i!=1) { i--,--i; cout<<11;} să afișeze 11111?**

- a) ciclează la infinit b) 5 c) 7 d) nu există o valoare

6. Dacă *x* și *i* sunt două variabile întregi, care dintre următoarele secvențe de instrucțiuni afișează numărul 11?

- a) *x=7;*
 for (i=1; i<4; i++, x++); cout<<x;
b) *x=1;*
 while (x<=11) x++; cout<<x;
c) *x=1;*
 for (i=1; i<=2; i++) cout<<x;
d) *x=1; i=2;*
 while (i>0) { cout<<x; i--; }

7. Care dintre următoarele secvențe de instrucțiuni atribuie variabilei întregi *y* valoarea cifrei celei mai semnificative a numărului natural reprezentat de variabila *x*?

- a) **for (y=x; y; y /=10);**
b) *y=x;*
 while (y>=0) y /=10;
c) **for (y=x; y>10; y /=10);**
d) **while (x>0) x /= 10;**
 y=x;

8. Care dintre următoarele secvențe de instrucțiuni atribuie variabilei întregi *x* valoarea n^2 , cu *n* număr natural, iar variabila *i* de tip întreg?

- a) *x=n;*
 for (i=1; i==2; i++) x *= n;
c) *x=n;*
 for (i=1; i< 2; i++); x *= n;

acțiunea while

- b) `x=1; i=0;`
 while (`i<2`) `x:= x * n; i++;`
- d) `x= 1;`
 for (`i=1; i<= 2; i++`) `x *= n;`
9. Care dintre următoarele secvențe afișează cel mai mare număr natural care este mai mic sau egal cu valoarea variabilei reale pozitive x ?
- a) `n= 0;`
 while (`n<=x`) `n++; cout<<n-1;`
- c) `n= 0; do {n=++} while (n<=x)`
 cout<<n-1;
- b) `n= 0; do { n++; } while (n<=x);`
 cout<<n;
- d) **for** (`n=1; n<=x; n++`);
 cout<<n-1;
10. Care dintre următoarele secvențe de instrucțiuni atribuie variabilei întregi x valoarea 10^n , cu n număr natural și variabila i de tip întreg?
- a) `x= 1;`
 for (`i=1; i< n; i++`) `x *= 10;`
- c) `x=1; i=0;`
 while (`i<n`) `x*= 10; i++;`
- b) `x= 10;`
 for (`i=1; i<= n; i++`) `x *= 10;`
- d) `x=1;`
 for (`i=1; i<= n; i++`) `x *= 10;`

Rezolvați:

- Se citește un număr care reprezintă luna din an (1 pentru ianuarie, 2 pentru februarie etc.). Afișați numărul lunii precedente și numărul lunii următoare.
- Se citesc două numere de la tastatură, care reprezintă anul și luna. Afișați numărul de zile din luna respectivă (pentru luna februarie se va ține cont de an: dacă este bisect sau nu).
- Se citesc mai multe cifre de la tastatură până când suma lor depășește valoarea 40. Afișați numărul de cifre introduse. Scrieți câte o variantă de program pentru fiecare structură repetitivă implementată în limbajul C++. Precizați expresiile și instrucțiunile folosite pentru cele trei acțiuni ale procesului de control.
- Se citește un număr natural n și apoi un șir de n numere întregi. Afișați mediile aritmetice ale tripletelor de numere pozitive introduse consecutiv. Scrieți câte o variantă de program pentru fiecare structură repetitivă implementată în limbajul C++. Precizați expresiile și instrucțiunile folosite pentru cele trei acțiuni ale procesului de control.
- Se citesc două numere de la tastatură, a și b . Afișați numărul de termeni ai șirului lui Fibonacci din intervalul $[a, b]$. Scrieți câte o variantă de program pentru fiecare structură repetitivă implementată în limbajul C++. Precizați expresiile și instrucțiunile folosite pentru cele trei acțiuni ale procesului de control.
- Să se afișeze toate numerele de forma $abba$ divizibile cu n (n se citește de la tastatură). Scrieți câte o variantă de program pentru fiecare structură repetitivă implementată în limbajul C++. Precizați expresiile și instrucțiunile folosite pentru cele trei acțiuni ale procesului de control.
- Să se calculeze produsul $a \times b$ a două numere întregi, fără să se folosească operatorul pentru înmulțire (se folosește adunarea repetată a lui a , de b ori, și se ține cont de semnul numerelor).
- Să se calculeze câtul și restul împărțirii a două numere întregi a și b , fără să se folosească operatorii $/$ și $\%$ (se folosește scăderea repetată a lui b din a și se ține cont de semnul numerelor).

9. Să se afișeze cuburile perfecte mai mici decât un număr n citit de la tastatură.
10. Să se afișeze numărul pătratelor perfecte mai mici decât un număr n citit de la tastatură.
11. Se citește un număr natural n și apoi un șir de n numere întregi. Afișați suma care o obțineți adunând primul divizor prim din fiecare număr citit.
12. Se citește un număr natural n și apoi un șir de n numere întregi. Afișați primul număr care are cei mai mulți divizori.
13. Se citesc mai multe numere naturale până se citește un număr negativ. Afișați ultimul număr care are cei mai puțini divizori primi.
14. Se citește un număr natural nenul care poate avea maxim 9 cifre. Afișați cifrele distincte ale numărului.
15. Se citește un număr natural nenul care poate avea maxim 9 cifre. Determinați dacă numărul are cifrele ordonate strict crescător sau strict descrescător.
16. Să se determine toate dubletele de numere întregi (x,y) care îndeplinesc condiția $x^2+y^2 = r^2$, unde r se citește de la tastatură.
17. Să se determine toate dubletele de numere întregi (x,y) care îndeplinesc condiția $x^2+y^2 < r^2$, unde r se citește de la tastatură.
18. Să se afișeze toate numerele de forma a^2+b^4 , cu $1 \leq a \leq 10$ și $1 \leq b \leq 10$.
19. Să se determine toate triunghiurile diferite care au lungimea laturilor numere naturale și perimetrul p (p se citește de la tastatură).
20. Se consideră numărul $n=2a^2+3b^3$, a și b fiind două numere naturale distincte cu $1 \leq a \leq 10$ și $1 \leq b \leq 10$. Să se afișeze toate numerele n generate astfel, și, pentru fiecare număr n generat, descompunerea lui în sumă de trei pătrate.
21. Să se determine toate dubletele de numere întregi (x,y) care îndeplinesc simultan condițiile: $-a < x < a$ și $-b < y < b$; a și b se citesc de la tastatură.
22. Să se determine toate tripletele de numere naturale (x,y,z) care îndeplinesc condiția $x^2+y^2+z^2=n$, unde n se citește de la tastatură.
23. Să se determine toate tripletele de numere naturale (x,y,z) care îndeplinesc simultan condițiile $1 \leq x \leq y \leq z \leq n$ și $x^2+y^2=z^2$; n se citește de la tastatură.
24. Să se determine toate cvadrupele de numere naturale (x,y,z,n) care îndeplinesc simultan condițiile: $n \geq 3$, $1 \leq x \leq y \leq z \leq m$ și $x^n+y^n=z^n$ unde m se citește de la tastatură.
25. Să se scrie toate modurile în care poate fi descompus un număr natural nenul în sumă de cuburi a două numere naturale. Dacă nu există nici un mod de descompunere, să se scrie mesajul "Imposibil".
26. Să se găsească cel mai mic număr natural nenul care poate fi descompus în sumă de cuburi a două numere naturale.
27. Să se găsească cel mai mic număr natural nenul care poate fi descompus în sumă de cuburi a două numere naturale, în cel puțin două moduri distincte.
28. Scrieți un program care să afișeze de câte ori apare o cifră nenulă c în scrierea tuturor numerelor naturale mai mici sau egale cu un număr dat n . Cifra c și valoarea lui n se citesc de la tastatură.

5. Implementarea structurilor de date

5.1. Structurile de date

Memoria internă a calculatorului este organizată ca un ansamblu de celule separate care au adrese consecutive. Într-o astfel de celulă sau într-un grup de celule adiacente, se poate memora o dată elementară. În același mod este organizată și memoria externă: un ansamblu de locații de memorare numite sectoare, care au adrese consecutive.

În cadrul unei aplicații pot să apară multe date de același tip. De exemplu, o firmă vrea să prelucreze cu ajutorul calculatorului situația vânzărilor. Informația prelucrată va fi organizată logic într-un tabel în care fiecare produs reprezintă un rând și fiecare zi dintr-o anumită perioadă (săptămână, lună, an etc.) reprezintă o coloană. Acest mod de organizare permite o analiză rapidă a informației și ușurează munca de obținere a informațiilor statistice (volumul vânzărilor dintr-o zi, volumul vânzărilor pentru un produs într-o anumită perioadă, cel mai bine vândut produs într-o anumită perioadă etc.). Organizarea logică a informațiilor în exteriorul calculatorului trebuie să se regăsească și în interiorul lui, în modul în care sunt organizate datele sub formă de colecții de date care reprezintă informațiile. Utilizatorul unei aplicații va gândi natural, fără să fie preocupat de modul în care sunt organizate efectiv datele în memoria calculatorului. Așadar, colecția de date sau structura de date reprezintă o metodă de aranjare a datelor care sunt dependente unele de altele în cadrul unei aplicații.

Studiu de caz

Scop: exemplificarea necesității folosirii unei structuri de date pentru rezolvarea unei probleme.

Enunțul problemei: Să se calculeze mediile generale ale elevilor unei clase.

Procesul de calculare a mediilor generale ale elevilor unei clase presupune ca, pentru fiecare elev, să se calculeze mediile anuale la fiecare disciplină și apoi să se calculeze media generală pe an. Să presupunem că există 15 discipline. Procesul constă în executarea următorului algoritm:

- Pasul 1.** Pentru fiecare disciplină se adună cele două medii semestriale și se împarte la 2 pentru a se obține media anuală.
- Pasul 2.** Se adună toate mediile anuale și rezultatul se împarte la numărul de discipline, adică 15, pentru a se obține media generală.

Această secvență de operații se va executa repetat pentru fiecare elev din clasă. Datele necesare prelucrării sunt:

- ✓ **Informațiile de intrare** sunt mediile semestriale la fiecare disciplină ale elevilor din clasă. Înseamnă că pentru fiecare elev din clasă va trebui să existe un set de **date de intrare** format din mediile semestriale la fiecare disciplină ($2 \times 15 = 30$ date de intrare).
- ✓ **Informația de ieșire** sunt mediile anuale la fiecare disciplină și media generală a elevilor din clasă. Înseamnă că pentru fiecare elev din clasă va trebui să existe un set de **date de ieșire** format din mediile anuale la fiecare disciplină și media generală ($15 + 1 = 16$ date de ieșire).

În total, pentru fiecare elev ar fi necesare 46 de date elementare. Dacă la acestea mai adăugăm cel puțin o dată elementară necesară pentru identificarea elevului în clasă sunt 25 de elevi, ar fi necesare $25 \times 47 = 1175$ de date elementare. Procesul de prelucrare a acestor date elementare presupune stabilirea a 1175 de tripluri pentru descrierea lor, inclusiv definirea a 1175 de identificatori diferiți între ei. Într-un astfel de proces de prelucrare a datelor este mult mai convenabil ca datelor să li se asociază un model de organizare sub forma unui tabel, în care fiecărui elev îi corespunde un rând al tabelului, iar fiecărei medii, o coloană. Programatorul va degreva de modul în care vor fi aranjate toate aceste date în memoria internă și în mecanismul de identificare a celor 1175 de date elementare. Prin acest model de organizare a datelor se construiește de fapt o structură de date.



Structura de date este o colecție de date între elementele căreia se definește un anumit tip de relație care determină metodele de localizare și prelucrare a datelor.

Așadar, structura de date este o metodă de aranjare a datelor care sunt dependente unele de altele în cadrul unei aplicații. Ea este o colecție de elemente pentru care s-a precizat:

- ✓ **tipul elementelor,**
- ✓ **proprietățile de organizare a elementelor,**
- ✓ **regulile de acces la elemente.**

Componentele unei structuri de date pot fi:

- ✓ **date elementare,**
- ✓ **structuri de date.**

O structură de date este o entitate de sine stătătoare. Ea **poate fi identificată printr-un nume**, iar componentele ei își mențin atributele. Fiecărei structuri de date îi este specific un anumit mecanism de identificare și de selecție a componentelor colecției de date.

Componentele structurii de date pot fi identificate prin:

- ✓ **identificatorul (numele) componentei, sau**
- ✓ **poziția pe care o ocupă componenta în cadrul structurii.**

...e elev din clasă.

...disciplină ale elevilor
...să existe un set de
...cină ($2 \times 15 = 30$ date

...media generală a
...trebuie să existe un
...disciplină și media

...Dacă la acestea
...ficarea elevului, iar
...mentare. Procesul
...1175 de triplete
...între ei. Într-un
...ca datelor să li se
...are fiecărui elev îi
...Programatorul va fi
...memoria internă și de
...Prin acest
...ze date.



...căreia se
...localizare și

...care sunt depen-
...elemente pentru

...ate fi identificată
...zării structuri de
...selecție a compo-

Structurile de date pot fi clasificate după diferite criterii:

1. în funcție de tipul componentelor structurii:
 - ✓ **structuri omogene** (componentele sunt de același tip),
 - ✓ **structuri neomogene** (componentele sunt de tipuri diferite);
2. în funcție de modul de localizare a componentelor structurii:
 - ✓ **structuri cu acces direct** (o componentă poate fi localizată fără să se țină cont de celelalte componente ale structurii),
 - ✓ **structuri cu acces secvențial** (o componentă poate fi localizată numai dacă se parcurg componentele care o preced în structură);
3. în funcție de tipul de memorie în care sunt create:
 - ✓ **structuri interne** (în memoria internă),
 - ✓ **structuri externe** (în memoria externă);
4. în funcție de timpul de utilizare:
 - ✓ **structuri de date temporare** (pot fi organizate atât în memoria internă, cât și în cea externă),
 - ✓ **structuri de date permanente** (pot fi organizate numai în memoria externă);
5. în funcție de stabilitatea structurii:
 - ✓ **structuri dinamice** (în timpul existenței, în urma executării unor procese, își modifică numărul de componente și relațiile dintre ele),
 - ✓ **structuri statice** (nu își modifică în timpul existenței numărul de componente și relațiile dintre ele).

Asupra unei structuri de date se pot executa mai multe operații care pot afecta valorile componentelor structurii și/sau structura de date:

1. **Crearea**, prin care se realizează structura de date în forma inițială, pe suportul de memorare utilizat.
2. **Consultarea**, prin care se realizează accesul la componentele structurii în vederea prelucrării valorilor acestora și a extragerii de informații.
3. **Actualizarea**, prin care se schimbă starea structurii astfel încât ea să reflecte corect valoarea componentelor la un moment dat. Actualizarea se face prin trei operații: **adăugarea** unor noi componente, **ștergerea** unor componente și **modificarea** valorii componentelor.
4. **Sortarea**, prin care se rearanjează componentele structurii în funcție de anumite criterii de ordonare aplicate valorilor componentelor.
5. **Copierea**, prin care se realizează o imagine identică a structurii, pe același suport sau pe suporturi diferite de memorare.
6. **Mutarea**, prin care se transferă structura, pe același suport, la o altă adresă, sau pe un suport de memorare diferit.
7. **Redenumirea**, prin care se schimbă numele structurii.
8. **Divizarea**, prin care se realizează două sau mai multe structuri dintr-o structură de bază.
9. **Reuniunea (concatenarea)**, prin care se realizează o singură structură de date, prin combinarea a două sau mai multe structuri de date de același tip.
10. **Ștergerea**, prin care se distruge structura de date.

Implementarea unei structuri de date presupune:

- ✓ **Definirea structurii din punct de vedere logic**, adică definirea componentelor, a relației dintre componente și a operațiilor care pot acționa asupra structurii.
- ✓ **Definirea structurii din punct de vedere fizic**, adică a modului în care va fi reprezentată structura pe suportul de memorare.

Se pot folosi mai multe tipuri de structuri de date.

Tipul de structură de date definește apartenența structurii de date la o anumită familie de structuri cărora le corespunde același mod de organizare logică, același model de reprezentare fizică și care pot fi supuse acelorași operații.

Dintre tipurile de structuri de date fac parte:

- ✓ **tablourile de memorie;**
- ✓ **fișierele.**

5.2. Tablourile de memorie

Să presupunem că o aplicație trebuie să ordoneze crescător un șir oarecare de 20 de numere. Cele 20 de numere sunt date de intrare și, la primul pas al algoritmului, vor fi introduse în memoria internă a calculatorului. La pasul următor numerele vor fi prelucrate și rearanjate conform criteriului de ordonare precizat, după care, la al treilea pas, vor fi furnizate utilizatorului sub formă de date de ieșire. În memoria internă ele vor fi înregistrate într-o structură de date de tip tablou de memorie.

Tabloul de memorie (array) este o structură de date internă formată dintr-o mulțime ordonată de elemente, ordonarea făcându-se cu un ansamblu de indici.

Tabloul de memorie se va identifica după un nume (a), iar fiecare element al său după numele tabloului și numărul de ordine. În cazul exemplului prezentat, cele 20 de numere se vor păstra într-o zonă continuă de memorie internă, care va conține 20 de celule de memorare. Presupunând că datele elementare care compun structura sunt de tip `int` și ocupă fiecare dintre ele câte doi octeți, înseamnă că întreaga structură va ocupa 40 de octeți. În loc să se atribuie datelor 20 de nume, se va atribui un singur nume întregii structuri, iar fiecare dată se va regăsi, în cadrul structurii, după numărul de ordine. Presupunând că adresa de la care începe structura de date este 500, aceasta reprezintă adresa primului element, adresa ultimului element fiind 538. După prelucrarea datelor de intrare prin procesul de sortare, se va obține un nou tablou de memorie (b) care conține datele de ieșire.

Fiecare element al structurii de date se identifică după numele tabloului și poziția din tablou. Astfel, elementul a_3 este al treilea element din tabloul a , iar data memorată are valoarea 1. Elementul b_2 este al doilea element din tabloul b , iar data memorată are valoarea 4. Se observă că, de la început, trebuie să se precizeze câte elemente va

structurilor de date

definirea componen-
ționale asupra struc-
turii în care va fi re-

de date la o anu-
organizare logică,
ceorași operații.

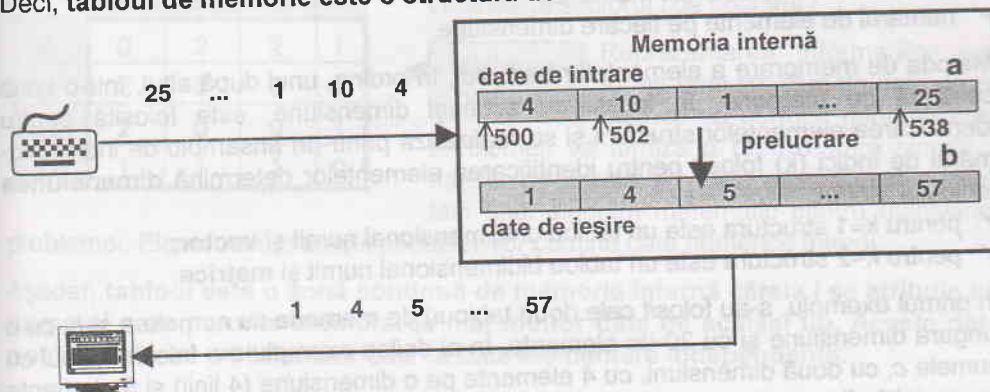
șir oarecare de 20
pas al algoritmului,
rător numerele vor fi
după care, la al
ieșire. În memoria
de memorie.

ă formată dintr-o
ansamblu de indici.

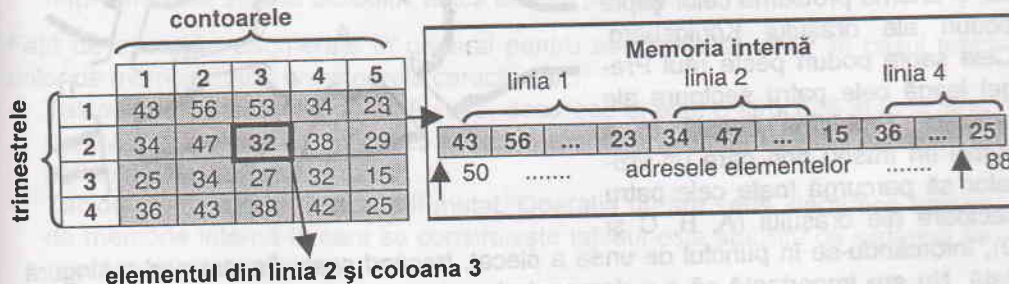
ecare element al său,
prezentat, cele 20
emă, care va conține
care compun struc-
seamnă că întreaga
de nume, se va atri-
în cadrul structurii,
se începe structura de
sa ultimului element
sortare, se va obține

tabloului și poziția din
data memorată are
data memorată are
câte elemente va

conține structura de date pentru ca sistemul să-i aloce zona de memorie corespunzătoare. În exemplu, se va preciza că cele două tablouri de memorie vor avea fiecare câte 20 de elemente de tip întreg pe 2 octeți, iar sistemul le va aloca fiecăruia dintre ele o zonă de memorie de 40 de octeți, așa cum alocă zonă de memorie internă și datelor elementare în funcție de tipul lor. Rezultă că, în timpul prelucrării, structura de date de tip tablou nu permite modificarea lungimii structurii, deoarece prin adăugarea de noi elemente se iese din spațiul de memorare alocat. Deci, **tabloul de memorie este o structură de date statică.**



Să considerăm o altă aplicație. Sunt cinci contoare pentru energia electrică, ce se citesc trimestrial, și se înregistrează într-un tabel consumul trimestrial (în mii de kWh). Tabelul va avea 4 linii, corespunzătoare celor 4 trimestre, și 5 coloane, corespunzător celor cinci contoare. Aceste date de intrare pot fi prelucrate prin diferite procese, obținându-se informații despre: media consumului trimestrial pe un singur contor sau pe toate contoarele, cel mai mare consum trimestrial, cel mai mic consum trimestrial, contorul cu cel mai mic consum etc. În vederea prelucrării, datele de intrare vor fi organizate într-o structură de date de tip tablou bidimensional cu patru linii și cinci coloane, cu numele *c*. Identificarea unui element din tablou se va face de data aceasta nu printr-un număr de ordine, ci prin numărul liniei și numărul coloanei. Astfel, elementul $c_{2,3}$ este elementul din linia 2 și coloana 3 și, în exemplul nostru, are valoarea 32.



Memoria internă nu are o structură matriceală. Ea este formată din locații de memorare care au adrese consecutive. Din această cauză, structura dreptunghiulară a

tabelului trebuie simulată. Spațiul de memorare alocat tabelului va fi o zonă continuă, de 40 de octeți, în care se vor memora datele din tabel, linie cu linie.

Numărul de ordine m al elementului din linia i și coloana j dintr-un tablou cu n coloane este:

$$m = n \times (i-1) + j$$

La definirea unui tablou de memorie trebuie specificate:

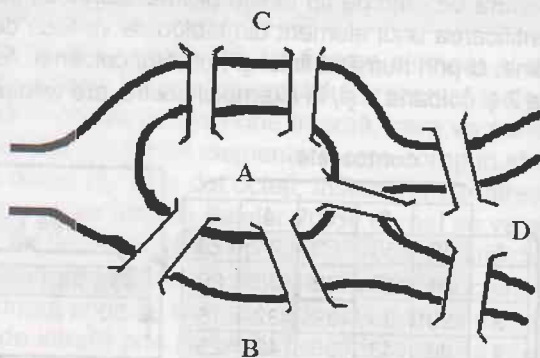
- ✓ numele tabloului de memorie,
- ✓ dimensiunea tabloului de memorie,
- ✓ numărul de elemente pe fiecare dimensiune.

Metoda de memorare a elementelor tabloului, în ordine, unul după altul, într-o zonă continuă de memorie, în locații cu aceeași dimensiune, este folosită pentru identificarea elementelor structurii și se realizează printr-un ansamblu de indici. Numărul de indici (k) folosit pentru identificarea elementelor determină **dimensiunea** tabloului. Astfel:

- ✓ pentru $k=1$ structura este un tablou unidimensional numit și **vector**;
- ✓ pentru $k=2$ structura este un tablou bidimensional numit și **matrice**.

În primul exemplu, s-au folosit cele două tablouri de memorie cu numele a și b , cu o singură dimensiune și cu 20 de elemente. În al doilea exemplu s-a folosit tabloul cu numele c , cu două dimensiuni, cu 4 elemente pe o dimensiune (4 linii) și 5 elemente pe cealaltă dimensiune (5 coloane). Și în cel de al doilea caz, tabloul are tot 20 de elemente. Chiar dacă, din punct de vedere al memoriei interne, spațiul de memorare alocat este tot sub forma unui bloc continuu de 40 de octeți, cele două tipuri de tablouri (cu o dimensiune sau cu două dimensiuni) diferă între ele prin modul în care pot fi identificate elementele structurii. Identificarea unui element al tabloului se face prin numele tabloului și valorile indicilor după toate dimensiunile. De exemplu, a_6 înseamnă elementul 6 din tabloul a , iar $c_{2,4}$ înseamnă elementul de pe linia 2 și coloana 4 din tabloul c .

Tablourile de memorie se pot folosi pentru a crea colecții de date care vor fi prelucrate de algoritmi. Să considerăm o problemă clasică, și anume problema celor șapte poduri ale orașului Königsberg. Cele șapte poduri peste râul Pregel leagă cele patru sectoare ale orașului. Problema constă în a stabili un traseu prin care un vizitator să parcurgă toate cele patru sectoare ale orașului (A, B, C și



D), întorcându-se în punctul de unde a plecat, trecând peste fiecare pod o singură dată. Nu are importanță că s-a demonstrat matematic că problema nu are soluții. Ceea ce ne interesează este modul în care pot fi furnizate aceste date algoritmului prin care trebuie să se găsească soluția traseului. Modul în care podurile leagă cele patru sectoare ale orașului poate fi reprezentat prin următoarea diagramă:

va fi o zonă conti-
nuă linie.

un tablou cu n

altul, într-o zonă
folosită pentru
de indici. Nu-
dimensiunea

ector;
matrice.

numele a și b , cu o
folosit tabloul cu
linii) și 5 elemente
tabloul are tot 20 de
de memorare
cele două tipuri de
prin modul în care
tabloului se face
De exemplu, a_6
de pe linia 2 și



care pod o singură
nu are soluții.
date algoritmului
podurile leagă cele
diagramă:

Această informație, prezentată acum sub formă de diagramă, poate fi reprezentată în memoria internă sub forma unui tablou bidimensional R , numit *matricea podurilor*. El are patru linii și patru coloane, care corespund celor patru sectoare ale orașului. Elementul

	A	B	C	D
A	0	2	2	1
B	2	0	0	1
C	2	0	0	1
D	1	1	1	0

Matricea podurilor

Reprezentarea informațiilor sub forma unei structuri de date de tip matrice va permite generalizarea problemei, va furniza o reprezentare simplă a datelor de intrare și va da posibilitatea dezvoltării unui algoritm matematic pentru rezolvarea

problemei. Elementele matricei podurilor vor conține date numerice întregi.

Așadar, **tabloul este o zonă continuă de memorie internă căreia i se atribuie un nume și care permite memorarea mai multor date de același tip. Aceste date pot fi tratate ca un tot unitar sau ca date elementare independente.**

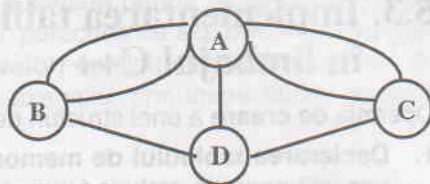
Tabloul de date este o structură de date omogenă, cu acces direct, internă, temporară, statică și liniară.

Așadar, implementarea unui tabel de memorie se face:

- ✓ **Din punct de vedere logic.** Tabloul de memorie este o structură de date omogenă, cu elemente de același tip. Este o structură liniară în care fiecare element, cu excepția ultimului element, are un succesor unic, localizarea unui element în cadrul structurii făcându-se cu ajutorul unui sistem de indici.
- ✓ **Din punct de vedere fizic.** Tabloului de memorie i se alocă o zonă continuă de memorie, de dimensiune fixă, care este împărțită în locații de aceeași dimensiune. În fiecare locație se memorează un element al tabloului. Dimensiunea unei locații este dată de tipul de dată memorată. Localizarea unui element al tabloului se face prin calcularea adresei elementului față de un element de referință care este adresa tabloului, adică adresa primului element.

Față de operațiile enumerate în general pentru structurile de date, în cazul tablourilor de memorie apar următoarele caracteristici:

- ✓ Tabloul de memorie nu poate fi șters, deoarece el este o structură statică temporară. La terminarea aplicației, zona de memorie alocată lui este eliberată și atribuită altei aplicații.
- ✓ Tabloul de memorie nu poate fi mutat. Operația nu are sens, deoarece adresa de memorie internă la care se construiește tabloul este stabilită de sistemul de operare, și nu de utilizator.
- ✓ Nu există operația de redenumire.



5.3. Implementarea tablourilor de memorie în limbajul C++

Operația de **creare** a unei structuri de date de tip tablou de memorie presupune:

1. **Declararea tabloului de memorie** pentru a i se alocă spațiu de memorie. Prin această operație, trebuie furnizate următoarele informații:
 - ✓ numele tabloului care va fi folosit în expresii pentru a-l identifica;
 - ✓ tipul elementelor tabloului;
 - ✓ dimensiunea tabloului pentru a preciza numărul de indici folosiți pentru localizarea elementelor;
 - ✓ numărul de elemente ale tabloului pentru a se cunoaște spațiul de memorie care trebuie alocat tabelului.
2. **Atribuirea de valori elementelor** tabloului, care se poate face prin mai multe metode:
 - ✓ prin inițializarea tabloului de memorie, la declararea lui;
 - ✓ prin introducerea valorilor de la tastatură;
 - ✓ prin preluarea valorilor dintr-o altă structură de date;
 - ✓ prin generarea valorilor folosind un algoritm de calcul al lor.

5.3.1. Tabloul cu o singură dimensiune – vectorul

Declararea unui tablou de memorie de tip vector (cu o singură dimensiune) se face prin instrucțiunea:

```
tip_dată nume [nr_elemente];
```

unde **tip_dată** precizează tipul elementelor vectorului, **nume** este identificatorul vectorului, iar **nr_elemente** este o constantă întreagă care specifică numărul de elemente ale vectorului. De exemplu, prin instrucțiunile declarative:

```
int a[20];  
float b[10];  
char c[30];
```

se declară trei vectori: *a* cu 20 de elemente, de tip **int**, *b* cu 10 elemente, de tip **float** și *c* cu 30 de elemente, de tip **char**.

Observație: Deoarece **nr_elemente** este o constantă întreagă, precizarea valorii sale se poate face fie printr-o constantă literală, fie printr-o constantă simbolică. De exemplu, prin instrucțiunile declarative:

```
const int DIM=50;  
int a[DIM];
```

se declară un vector *a* cu 50 de elemente de tip **int**. Avantajul folosirii constantei simbolice este acela că aveți mecanismul prin care să folosiți în program informația despre numărul de elemente ale vectorului.

La declararea unui vector se pot atribui valori inițiale elementelor, cu instrucțiunea:

```
tip_dată nume [nr_elemente] = {listă_valori};
```

unde caracterul separator = este folosit pentru a separa partea de declarare a vectorului de partea de inițializare a elementelor lui, perechea de separatori {...} se folosește pentru a delimita **listă_valori**, iar **listă_valori** conține valorile inițiale ale elementelor, separate prin caracterul virgulă. De exemplu, prin instrucțiunile declarative:

```
int a[5] = {10,20,30,40,50};
float b[4] = {0.1, 0.2, 0.3, 0.4};
```

se declară doi vectori cu valori inițiale pentru elemente:

a cu 5 elemente de tip **int**

10	20	30	40	50
----	----	----	----	----

indici 0 1 2 3 4

b cu 4 elemente de tip **float**

0.1	0.2	0.3	0.4
-----	-----	-----	-----

indici 0 1 2 3

În cazul declarării unui vector inițializat, se poate omite numărul de elemente al vectorului, dacă se inițializează toate elementele. Compilatorul va considera că vectorul are atâtea elemente, câte valori sunt în lista de valori. De exemplu, prin instrucțiunile declarative:

```
int a[] = {10,20,30,40,50};
float b[] = {0.1, 0.2, 0.3, 0.4};
```

se declară doi vectori: **a** cu 5 elemente, de tip **int** și **b** cu 4 elemente, de tip **float**.

Dacă lista de valori conține mai puține valori decât elementele vectorului, restul elementelor vor fi inițializate cu valoarea 0. De exemplu, prin instrucțiunile declarative:

```
int a[5] = {10,20};
float b[4] = {0.1, 0.2, 0.3};
```

se declară doi vectori, cu valori inițiale pentru elemente:

a cu 5 elemente de tip **int**

10	20	0	0	0
----	----	---	---	---

indici 0 1 2 3 4

b cu 4 elemente de tip **float**

0.1	0.2	0.3	0
-----	-----	-----	---

indici 0 1 2 3

Inițializarea unui vector de caractere se poate face fie prin atribuirea unei constante de tip caracter fiecărui element, fie prin atribuirea unei constante de tip șir de caractere, vectorului. De exemplu, prin instrucțiunile declarative:

```
char c[5] = {'a', 'b', 'c', 'd', '\0'};
char d[5] = "abcd";
```

se declară doi vectori de caractere care au aceleași valori inițiale pentru elemente:

c cu 5 elemente de tip **char**

a	b	c	d	nul
---	---	---	---	-----

indici 0 1 2 3 4

c cu 5 elemente de tip **char**

a	b	c	d	nul
---	---	---	---	-----

indici 0 1 2 3 4

Caracterul **null** (caracterul care are codul ASCII 0: '\0') este folosit ca un terminator al șirului de caractere și este utilizat pentru a ușura prelucrarea vectorilor cu șiruri de caractere. Și în cazul vectorilor de caractere este posibil să nu se declare nu-

mărul de elemente ale vectorului, dacă se inițializează toate elementele vectorului. Instrucțiunile declarative:

```
char c[] = {'a', 'b', 'c', 'd', '\0'};
```

```
char d[] = "abcd";
```

sunt echivalente cu cele anterioare.

Datele memorate în vector vor putea fi folosite pentru a fi prelucrate. Pentru prelucrare, se vor construi expresii în care elementele vectorului vor fi operanzi.

Referirea la un element al vectorului se face prin construcția:

```
nume[indice]
```

unde **nume** este numele vectorului, **indice** este numărul de ordine al elementului în cadrul vectorului, iar perechea de operatori [...] se folosește pentru a calcula adresa elementului față de un element de referință care este adresa vectorului. Prioritatea operatorilor [...] este prioritatea maximă (la fel ca și a funcțiilor).



Deoarece adresa vectorului este și adresa primului element, iar indicele reprezintă deplasarea elementului față de adresa vectorului, în limbajul C++ numerotarea indicilor se face pornind de la 0, și $0 \leq \text{indice} < n$, unde n reprezintă numărul de elemente ale vectorului, precizate la declararea lui.

De exemplu, $a[3]$ reprezintă al patrulea element din vectorul a și are valoarea 40, iar $b[2]$ reprezintă al treilea element din vectorul b și are valoarea 0.3.

5.3.2. Tabloul cu două dimensiuni – matricea

Declararea unui tablou de memorie de tip matrice (cu două dimensiuni) se face prin instrucțiunea:

```
tip_dată nume[nr_1][nr_2];
```

unde **tip_dată** precizează tipul elementelor matricei, **nume** este identificatorul matricei, iar **nr_1** și **nr_2** sunt două constante întregi care specifică numărul de elemente ale matricei pentru fiecare dimensiune: **nr_1** – numărul de linii, iar **nr_2** – numărul de coloane. De exemplu, prin instrucțiunile declarative:

```
int a[2][4];
```

```
float b[3][5];
```

se declară două matrice: a cu 8 elemente de tip **int**, care are 2 linii și 4 coloane, și b , cu 15 elemente de tip **float**, care are 3 linii și 5 coloane.

Și la declararea unei matrice se pot atribui valori inițiale elementelor, cu instrucțiunea:

```
tip_dată nume[nr_1][nr_2] = {listă_valori};
```

unde valorile din listă se vor atribui elementelor ținând cont de faptul că memorarea lor în zona alocată matricei se face linie după linie. De exemplu, prin instrucțiunea declarativă:

```
int a[2][4] = {1,2,3,4,5,6,7,8}; sau
```

```
int a[2][4] = {{1,2,3,4},{5,6,7,8}};
```

se declară o matrice cu valori inițiale, pentru elemente:

Pentru prelucrare a matricei se

unde **nume** este **indice_1** este



de linii și m
 $a[1][2]$ reprezintă
valoarea 7.



numărul de
lucrarea tabloului.

- ✓ **Lungimea** matricei. Este rezervat.
- ✓ **Lungimea** matricei. Este rezervat.

Pentru a verifica dacă matricea este pătrată sau eg

```
const int n=10;
int i,n,a[10][10];
// se declară matricea
// n este numărul de linii și coloane
cout<<"n="<<n<<"\n";
// se citesc valorile din matrice
while (n>0)
{
    cout<<"n="<<n<<"\n";
    cout<<"\n";
}
```

indicele coloanei j		0	1	2	3
indicele liniei i	0	1	2	3	4
	1	5	6	7	8

Pentru prelucrarea datelor memorate într-o matrice, **referirea la un element al matricei** se face prin construcția:

nume [indice_1] [indice_2]

unde **nume** este numele matricei, **indice_1** este numărul de ordine al liniei, iar **indice_2** este numărul de ordine al coloanei.

Atenție

Deoarece adresa matricei este și adresa primului element, iar indicii reprezintă deplasarea elementului față de adresa matricei, numerotarea indicilor se face pornind de la 0, iar $0 \leq \text{indice_1} < n$ și $0 \leq \text{indice_2} < m$, unde n reprezintă numărul de linii și m numărul de coloane ale matricei, precizate la declararea ei. De exemplu, $a[1][2]$ reprezintă elementul din linia a 2-a și coloana a 3-a din matricea a , și are valoarea 7.

Atenție

Tabloul de memorie fiind o structură de date statică, la declararea lui i se alocă o zonă fixă de memorie. **Lungimea tabloului de memorie** reprezintă **numărul de elemente** ale tabloului. La declararea tabloului este posibil să nu se cunoască numărul de elemente care vor fi prelucrate la fiecare execuție a programului. În prelucrarea tabloului de memorie se folosesc două lungimi:

- ✓ **Lungimea fizică.** Reprezintă numărul de elemente stabilit la declararea tabloului. Este numărul maxim de elemente care pot fi stocate în spațiul de memorie rezervat tabloului.
- ✓ **Lungimea logică.** Reprezintă numărul de elemente care vor fi prelucrate la execuția programului. Este mai mic sau cel mult egal cu lungimea fizică (numărul maxim de elemente). Lungimea logică se comunică în timpul execuției programului, de la tastatură. Ea reprezintă numărul de elemente ale vectorului, respectiv numărul de linii și numărul de coloane ale matricei.

Pentru a verifica dacă lungimea logică a vectorului (n) citită de la tastatură este mai mică sau egală cu lungimea fizică, puteți folosi următoarea secvență de instrucțiuni:

```
const int DIM=50; // se declară constanta numerică DIM
int i,n,a[DIM];
// se declară variabilele i pentru indicele elementului,
// n pentru lungimea logică a vectorului și a pentru un
// vector cu lungimea fizică de 50 de elemente
cout<<"n= "; cin>>n;
// se citește lungimea logică a vectorului
while (n>DIM)
{cout<<"Lungimea maximă a vectorului este "<<DIM<<endl;
  cout<<"Introduceți corect lungimea vectorului<<endl;
  cout<<"n= "; cin>>n;}
```


5.4. Algoritmi pentru prelucrarea tablourilor de memorie

Algoritmii pentru prelucrarea tablourilor de memorie sunt:

- ✓ algoritmi pentru parcurgerea elementelor tabloului de memorie,
- ✓ algoritmi pentru căutarea unui element în tabloul de memorie,
- ✓ algoritmul pentru ștergerea unui element dintr-un vector,
- ✓ algoritmul pentru inserarea unui element într-un vector,
- ✓ algoritmul pentru sortarea elementelor unui vector,
- ✓ algoritmul pentru interclasarea a doi vectori sortați.

5.4.1. Algoritmi pentru parcurgerea elementelor tabloului de memorie

Parcurgerea elementelor unui tablou de memorie înseamnă vizitarea pe rând a elementelor tabloului, în vederea executării unor prelucrări cu elementele tabloului:

- ✓ atribuirea de valori elementelor,
- ✓ preluarea valorii elementelor în vederea executării unor calcule necesare pentru rezolvarea problemei,
- ✓ afișarea valorii elementelor.

Parcurgerea elementelor tabloului se face secvențial, în ordinea indicilor. Ea se poate face de la primul element până la ultimul element, sau invers, de la ultimul element până la primul element. Parcurgerea elementelor tabloului este un proces repetitiv. Instrucțiunea cea mai adecvată pentru implementarea acestui algoritm este instrucțiunea **for**.

Algoritmi pentru parcurgerea elementelor unui vector

Parcurgerea unui vector se poate face:

- ✓ de la primul element până la ultimul element, sau
- ✓ de la ultimul element până la primul element.

Secvența de instrucțiuni pentru **parcurgerea elementelor unui vector** de la primul element la ultimul element este:

```
int i,n,a[10];
// se declară variabilele i pentru indicele elementului,
// n pentru lungimea logică a vectorului și a pentru un
// vector cu lungimea fizică de 10 elemente
cout<<"n= "; cin>>n;
// se citește lungimea logică a vectorului
for (i=0;i<n;i++) // se parcurg elementele vectorului
    .....;
// instrucțiunea pentru prelucrarea elementului a[i]
```

- Secvența de instrucțiuni pentru **parcurgerea elementelor unui vector** de la ultimul element la primul element este:

```
int i,n,a[10];
cout<<"n= "; cin>>n;
for (i=n-1;i>=0;i--) // se parcurg elementele vectorului
    .....;
// instrucțiunea pentru prelucrarea elementului a[i]
```

Algoritmi pentru parcurgerea elementelor unei matrice

Parcurgerea unei matrice se poate face:

- ✓ de la primul element până la ultimul element, sau
- ✓ de la ultimul element până la primul element.

Secvența de instrucțiuni pentru **parcurgerea elementelor unei matrice de la primul element la ultimul element** este:

```
int i,j,n,m,a[10][10];
// se declară variabilele i și j pentru indicele elementului,
// i - numărul liniei și j - numărul coloanei
// n și m pentru lungimea logică a matricei (n - numărul de
// linii și m - numărul de coloane) și a pentru o matrice cu
// lungimea fizică de 10 linii și 10 coloane.
cout<<"n= "; cin>>n;
// se citește numărul de linii ale matricei
cout<<"m= "; cin>>m;
// se citește numărul de coloane ale matricei
for (i=0;i<n;i++) // se parcurg liniile matricei
    for (j=0;j<m;j++) // se parcurg coloanele matricei de pe o linie
        .....;
// instrucțiunea pentru prelucrarea elementului a[i][j]
```

Secvența de instrucțiuni pentru **parcurgerea elementelor unei matrice de la ultimul element la primul element** este:

```
int i,j,n,m,a[10][10];
cout<<"n= "; cin>>n; cout<<"m= "; cin>>m;
for (i=n-1;i>=0;i--) // se parcurg liniile matricei
    for (j=m-1;j>=0;j--) // se parcurg coloanele matricei de pe o linie
        .....;
// instrucțiunea pentru prelucrarea elementului a[i][j]
```

Studiu de caz

Scop: exemplificarea modului în care se aplică algoritmi pentru parcurgerea tablourilor de memorie.

Enunțul problemei 1: Se citesc de la tastatură cel mult 10 numere întregi. Să se calculeze media aritmetică a numerelor strict pozitive. Pentru memorarea numerelor se va folosi o structură de date de tip vector.

Algoritmul de parcurgere se va folosi de două ori: o dată pentru a atribui elementelor valorile care se citesc de la tastatură (operația de **creare a vectorului**) și o dată

pentru calcularea sumei numerelor strict pozitive și numărarea lor (operația de **consultare a vectorului**). Parcurgerea vectorului se va face de la primul element până la ultimul element.

Exemplul 1

```
#include <iostream.h>
void main()
{int i,n,s,k,a[20];
// variabila s se folosește pentru suma numerelor strict
// pozitive, iar variabila k se folosește pentru numărarea
// elementelor strict pozitive
cout<<"n= "; cin>>n;
for (i=0;i<n;i++) // se parcurge vectorul pentru creare
// instrucțiunea pentru prelucrarea elementului a[i]
{cout<<"a["<<i+1<<"]= ";
// se afișează numărul de ordine al elementului, care este
// cu 1 mai mare decât indicele
cin>>a[i];}
for (i=0;i<n;i++) // se parcurge vectorul pentru consultare
// instrucțiunea pentru prelucrarea elementului a[i]
if (a[i]>0){s+=a[i]; k++;}
// dacă s-a găsit cel puțin un element strict pozitiv
// se calculează media și se afișează
if (k>0) cout<<"media= "<<(float)s/k;
}
```

Această problemă se poate rezolva și fără să se folosească un vector, folosind o singură variabilă de memorie pentru numerele citite. Folosirea în acest caz a vectorului ca metodă de stocare a datelor de intrare nici nu se recomandă. Această implementare a algoritmului nu este eficientă, deoarece se face risipă de o resursă a calculatorului: memoria internă. Exemplul următor necesită însă folosirea vectorilor.

Enunțul problemei 2: Se introduc de la tastatură cel mult 100 de numere întregi. Să se afișeze în ordine inversă numerele citite.

Algoritmul de parcurgere se va folosi de două ori: o dată pentru a atribui elementelor valorile care se citesc de la tastatură (operația de **creare a vectorului**) și o dată pentru afișarea elementelor vectorului *a* (operația de **consultare a vectorului**). La crearea vectorului, parcurgerea se va face de la primul element până la ultimul element, iar la consultare de la ultimul element până la primul element.

Exemplul 2

```
#include <iostream.h>
void main()
{int i,n,a[100];
cout<<"n= "; cin>>n;
for (i=0;i<n;i++) // se parcurge vectorul pentru creare
{cout<<"a["<<i+1<<"]= "; cin>>a[i];}
for (i=n-1;i>=0;i--) // se parcurge vectorul pentru consultare
cout<<a[i]<<" ";
}
```

Enunțul problemei 3: Se introduc de la tastatură cel mult 100 de numere întregi. Să se afișeze valoarea cea mai mică și numărul de ordine al elementelor care au valoarea minimă.

Pentru memorarea numerelor se va folosi un vector *a* cu lungimea fizică de 100 de elemente, iar pentru memorarea numărului de ordine al elementelor cu valoarea minimă se va folosi un vector *b*. Primul element al vectorului *b* se va folosi pentru memorarea minimului. Vectorul *b* va avea lungimea fizică de 101 elemente, deoarece este posibil ca toate elementele vectorului *a* să aibă valoarea minimă. Vectorul *b* se creează folosindu-se un algoritm de calcul (se atribuie valori rezultate în urma unui proces de analiză a elementului curent din vectorul *a*). Algoritmul de parcurgere se va folosi de trei ori: o dată pentru a atribui elementelor vectorului *a* valorile care se citesc de la tastatură (operația de **creare a vectorului a**), o dată pentru determinarea minimului și a pozițiilor în care se găsește (operația de **consultare a vectorului a** și de **creare a vectorului b**) și o dată pentru afișarea pozițiilor elementelor cu valoarea minimă din vectorul *a* (operația de **consultare a vectorului b**).

Exemplul 3

```
#include <iostream.h>
void main()
{int i,j,n,a[100],b[101];
// variabila i se folosește pentru parcurgerea vectorilor
// variabila j se folosește pentru indicele vectorului b
// atunci când se memorează în vector pozițiile
cout<<"n= "; cin>>n;
for (i=0;i<n;i++) {cout<<"a["<<i+1<<"]= "; cin>>a[i];}
b[0]=a[0];j=1;b[1]=1;
// se inițializează minimul cu valoarea primului element
// din vectorul a și se memorează în vectorul b poziția
// acestui element
for (i=1;i<n;i++)
if (a[i]<b[0]) //dacă elementul curent este mai mic decât
// valoarea minimă
{b[0]=a[i]; j=1; b[j]=i+1;}
// se atribuie minimului noua valoare și
// se inițializează cu 1 indicele vectorului b
// se atribuie elementului b[1] poziția minimului
else if (a[i]==b[0]) //dacă elementul curent este egal
//cu valoarea minimă
{j++; b[j]=i+1;}
//se incrementează indicele vectorului b cu 1
// se atribuie elementului b[j] poziția minimului
cout<<"minim= "<<b[0]<<endl; // se afișează minimul
cout<<"si s-a gasit in pozitiile:"<<endl;
for (i=1;i<=j;i++)
cout<<b[i]<<" ";
// se parcurge vectorul b pentru a afișa pozițiile minimului
}
```

Enunțul problemei 4: Se introduc de la tastatură cel mult 100 de numere întregi. Să se afișeze mai întâi numerele pare și apoi numerele impare.

Se vor folosi trei vectori: *a* pentru numerele citite, *b* pentru numerele pare și *c* pentru numerele impare. Toți vectorii vor avea lungimea fizică de 100 de elemente, deoarece este posibil ca toate numerele citite să fie numai pare sau numai impare. Se vor folosi pentru indicii vectorilor variabilele: *i* pentru vectorul *a*, *j* pentru vectorul *b* și *k* pentru vectorul *c*. Indicii *j* și *k* se inițializează cu 0. Ei au și funcția de contor (de a număra câte numere sunt pare și câte numere sunt impare). Vectorii *b* și *c* se creează prin preluarea valorilor dintr-un alt vector.

Exemplul 4

```
#include <iostream.h>
void main()
{int i,j=0,k=0,n,a[100],b[100],c[100];
 cout<<"n= "; cin>>n;
 for (i=0;i<n;i++) {cout<<"a["<<i+1<<"]= "; cin>>a[i];}
 for (i=0;i<n;i++)
    if (a[i]%2==0) {b[j]=a[i]; j++;}
    else {c[k]=a[i]; k++;}
 if (j)
    {cout<<"Numerele pare sunt:"<<endl;
     for (i=0;i<j;i++) cout<<b[i]<<" ";}
 else cout<<"Nu exista numere pare"<<endl;
 if (k)
    {cout<<"Numerele impare sunt:"<<endl;
     for (i=0;i<k;i++) cout<<c[i]<<" ";}
 else cout<<"Nu exista numere impare"<<endl;
 }
```

Enunțul problemei 5: Să se copieze vectorul *a* în vectorul *b*.



Pentru copierea unui vector, ca de exemplu **copierea vectorului *a* în vectorul *b***, nu se poate folosi operația de atribuire ***b=a***. Copierea se va face prin parcurgerea simultană a vectorilor *a* și *b*, folosind același indice *i*, și atribuirea elementului curent, din vectorul *b*, a valorii elementului curent din vectorul *a*.

Exemplul 5

```
#include <iostream.h>
void main()
{int i,n,a[100],b[100];
 cout<<"n= "; 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]<<" ";
 }
```

Enunțul problemei 6: Să se divizeze vectorul *a* (cu *n* elemente), în doi vectori, *b* și *c*. Primele *m* elemente din vectorul *a* se vor regăsi în vectorul *b*.

Exemplul 6

```
#include <iostream.h>
void main()
{int i,n,m,a[100],b[100],c[100];
 cout<<"n= "; cin>>n;
 cout<<"m= "; cin>>m;
```

se pare și c pentru
elemente, deoa-
mai impare. Se vor
vectorul b și k
de contor (de a
vectorii b și c se

```
for (i=0; i<n; i++) {cout<<"a["<<i+1<<"]= "; cin>>a[i];}
for (i=0; i<m; i++)
    b[i]= a[i]; // se creează vectorul b
for (i=m; i<n; i++)
    c[i-m]= a[i]; // se creează vectorul c
for (i=0; i<m; i++)
    cout<<b[i]<<" "; // se afișează vectorul b
for (i=0; i<n-m; i++)
    cout<<c[i]<<" "; // se afișează vectorul c
}
```

Enunțul problemei 7: Să se concateneze doi vectori, a cu n elemente, și b cu m elemente, în vectorul c.

Exemplul 7

```
#include <iostream.h>
void main()
{int i,n,m,a[100],b[100],c[200];
 cout<<"n= "; cin>>n;
 cout<<"m= "; cin>>m;
 for (i=0; i<n; i++) {cout<<"a["<<i+1<<"]= "; cin>>a[i];}
 for (i=0; i<m; i++) {cout<<"b["<<i+1<<"]= "; cin>>b[i];}
 for (i=0; i<n; i++)
    c[i]= a[i]; // se copiază în c vectorul a
 for (i=0; i<m; i++)
    c[n+i]= b[i]; // se adaugă la c vectorul b
 for (i=0; i<n+m; i++)
    cout<<c[i]<<" "; // se afișează vectorul c
}
```

Enunțul problemei 8: Să se inverseze un vector a, care are cel mult 100 de numere întregi.

Inversarea unui vector se poate face în două moduri:

- ✓ Vectorul inversat se creează într-un alt vector b.
- ✓ Vectorul se inversează în el însuși.

Inversarea unui vector în alt vector. Elementul a[0] se va atribui elementului b[n-1], elementul a[1] se va atribui elementului b[n-2], ..., elementul a[i] se va atribui elementului b[n-i-1], ..., elementul a[n-1] se va atribui elementului b[0]. Se observă că suma indicilor celor doi vectori este întotdeauna n-1.

Exemplul 8_1

```
#include <iostream.h>
void main()
{int i,n,a[100],b[100];
 cout<<"n= "; cin>>n;
 for (i=0; i<n; i++) {cout<<"a["<<i+1<<"]= "; cin>>a[i];}
 for (i=0; i<n; i++) b[i]=a[n-i-1];
 for (i=0; i<n; i++) cout<<b[i]<<" ";
}
```

- **Inversarea unui vector în el însuși.** Se va face prin interschimbarea elementelor simetrice: elementul a[0] se va interschimba cu elementul a[n-1], elementul a[1] se

va interschimba cu elementul $a[n-2]$, ..., elementul $a[i]$ se va interschimba cu elementul $a[n-i-1]$, ..., . Interschimbarea elementelor se va face până la jumătatea vectorului (elementul cu indicele $[n/2]$). Pentru interschimbare se va folosi o variabilă auxiliară x de același tip ca și elementele vectorului.

Exemplul 8_2

```
#include <iostream.h>
void main()
{int i,n,x,a[100];
  cout<<"n= "; cin>>n;
  for (i=0;i<n;i++) {cout<<"a["<<i+1<<"]= ";   cin>>a[i];}
  for (i=0;i<n/2;i++) {x=a[i]; a[i]=a[n-i-1]; a[n-i-1]=x;}
  for (i=0;i<n;i++) cout<<a[i]<<" ";
}
```

Enunțul problemei 9: Se citește un număr natural. Să se afișeze frecvența cifrelor sale (cifrele care apar în număr, și de câte ori apar).

Pentru memorarea cifrelor numărului n se va folosi vectorul a . Vectorul b are 10 elemente și se folosește pentru a număra de câte ori apare cifra i în număr (i reprezintă indicele elementului din vectorul b). Așadar, vectorul b este un vector de contoare, în care elementul cu indicele i conține numărul de apariții ale cifrei i . Elementele vectorului b se inițializează cu 0. Algoritmul este:

- Pas 1.** Se extrag cifrele din numărul n și se memorează în vectorul a .
- Pas 2.** Se parcurge vectorul a . Elementul curent al vectorului a este o cifră a numărului n și furnizează informații despre contorul care trebuie incrementat cu 1 (elementul din vectorul b care trebuie incrementat cu 1). Așadar, se incrementează cu 1 elementul din vectorul b care are ca indice valoarea elementului curent din vectorul a .
- Pas 3.** După parcurgerea vectorului a se vor afișa, din vectorul b , numai elementele diferite de 0 (contoarele diferite de 0 și care corespund cifrelor care au apărut cel puțin o dată în număr).

Exemplul 9

```
#include <iostream.h>
void main()
{unsigned long n;
  int i,j,a[10],b[10]={0};
  cout<<"n ="; cin>>n;
  i=0;
  while (n) {a[i]=n%10; n/=10; i++;}
  // i este lungimea logică a vectorului a
  for (j=0;j<i;j++) b[a[j]]++;
  for (i=0;i<=9;i++)
    if (b[i])
      cout<<"cifra "<<i<<" apare de "<<b[i]<<" ori  "<<endl;
}
```

Enunțul problemei 10: Să se calculeze suma elementelor de pe diagonala principală a unei matrice pătrate, de dimensiune n ($n \leq 10$). Elementele matricei sunt numere întregi.

se schimbă cu ele-
se jumătatea vecto-
se o variabilă auxili-

frecvența cifrelor

Vectorul b are 10
în număr (i repre-
este un vector de
aparitii ale cifrei i .

Vectorul a .
este o cifră a nu-
trebuie incrementat
1). Așadar, se
indice valoarea

numai elemen-
cifrelor care au

O **matrice pătrată** este o matrice în care numărul liniilor este egal cu numărul coloanelor. Elementele de pe diagonala principală au cei doi indici egali (numărul liniei este egal cu numărul coloanei).

Exemplul 10

```
#include <iostream.h>
void main()
{int n,i,j,s=0,a[10][10];
 cout<<"n="; cin>>n;
 for (i=0;i<n;i++) //se parcurge matricea pentru creare
   for (j=0;j<n;j++)
     {cout<<"a["<<i+1<<","<<j+1<<"]=" "; cin>>a[i][j];}
 for (i=0;i<n;i++) //se parcurge diagonala principală
   s+=a[i][i];
 cout<<"suma="<<s;
}
```

Enunțul problemei 11: Să se verifice dacă o matrice pătrată de dimensiune n ($n \leq 10$) are următoarea proprietate: elementele de pe diagonala secundară au valoarea 1, iar restul elementelor au valoarea 0. Elementele matricei sunt numere întregi.

Elementele de pe diagonala secundară au suma indicilor egală cu $n-1$ ($i+j=n-1$). Se va folosi o variabilă x , care va avea valoarea 1 dacă matricea are proprietatea specificată (valoarea logică *True*), și valoarea 0 dacă matricea nu are proprietatea specificată (valoarea logică *False*). Variabila x se inițializează cu 1 (presupunem că matricea are proprietatea specificată). Deoarece, pentru a determina proprietatea matricei, nu trebuie parcurse toate elementele, în cazul în care se găsește un element care nu îndeplinește condiția din proprietate parcurgerea matricei nu se va mai face cu o instrucțiune **for**, ci cu o instrucțiune **while**. Se vor parcurge coloanele unei linii prin incrementarea indicelui j , după care se va trece la linia următoare, prin incrementarea indicelui i și inițializarea cu 0 a indicelui j .

Exemplul 11

```
#include <iostream.h>
void main()
{int n,i,j,x=1,a[10][10];
 cout<<"n="; cin>>n;
 for (i=0;i<n;i++)
   for (j=0;j<n;j++)
     {cout<<"a["<<i+1<<","<<j+1<<"]=" "; cin>>a[i][j];}
 i=0; j=0;
 while (i<n && x)
   {if (i+j==n-1)
     {if (a[i][j]!=1) x=0;}
     else
       if (a[i][j]!=0) x=0;
     if (j==n-1){j=0; i++;}
     else j++;}
 if (x) cout<<"Matricea are proprietatile specificate";
 else
   cout<<"Matricea nu are proprietatile specificate"; }
```


Enunțul problemei 12: Să se inverseze coloanele unei matrice cu n linii și m coloane ($n \leq 10$, $m \leq 10$) cu elementele numere întregi.

Vom folosi algoritmul de inversare a unui vector în el însuși, pentru fiecare linie a matricei.

Exemplul 12

```
#include <iostream.h>
void main()
{int n,m,i,j,x,a[10][10];
 cout<<"n = "; cin>>n;
 cout<<"m = "; cin>>m;
 for (i=0;i<n;i++)
  for (j=0;j<m;j++)
   {cout<<"a["<<i+1<<" "<<j+1<<"]= "; cin>>a[i][j];}
 for (i=0;i<n;i++)
  for (j=0;j<m/2;j++)
   {x=a[i][j]; a[i][j]=a[i][m-j-1]; a[i][m-j-1]=x;}
 for (i=0;i<n;i++)
  {for (j=0;j<m;j++)cout<<a[i][j]<<" ";
   cout<<endl; }
}
```



5.4.2. Algoritmi pentru căutarea unui element într-un tablou de memorie

Acest algoritm găsește primul element din tablou a cărui valoare este o valoare precizată x , în vederea prelucrării acestui element prin una dintre operațiile următoare:

- ✓ modificarea valorii lui;
- ✓ ștergerea elementului din tablou;
- ✓ inserarea unui element după acest element sau înaintea lui;
- ✓ consultarea elementului în vederea efectuării unor calcule.

Pentru căutarea unui element într-un tablou de memorie se pot folosi doi algoritmi:

- ✓ algoritmul de căutare într-un tablou de memorie nesortat;
- ✓ algoritmul de căutare într-un tablou de memorie sortat.

Algoritmi pentru căutarea într-un tablou de memorie nesortat

În acest caz, căutarea elementului se face prin **parcurea secvențială** a tabloului de memorie până când este găsit elementul.

Secvența de instrucțiuni pentru **căutarea elementului într-un vector** este:

```
int i=0,n,x,a[10];
// se declară variabila x pentru valoarea elementului
cout<<"n= "; cin>>n;
cout<<"x= "; cin>>x;
..... // se creează vectorul
i=0; while (i<n && a[i]!=x) i++;
// sau for (i=0; i<n && a[i]!=x; i++);
```

```
// se
// el
if (i!
e.
```

Secvența

```
int i=
cout<<
cout<<
cout<<
.....
// se
// se
// pã
i=0; j
while
if (j
if (i!
cout<
else
cout<
```

Algoritmi

Unul din
Acest al
caută ele
indice al
rului (ind
vectorulu
mijloc. D
sunt ega
✓ subve
✓ subve

Așadar,
găsi eler
care se
elementu
seamnă

Se folose
✓ variab
✓ variab
✓ variab
✓ variab
✓ variab
✓ variab

```
// se parcurg elementele vectorului până se găsește
// elementul sau până se termină de parcurs tot vectorul
if (i!=n) cout<<"S-a gasit elementul in pozitia "<<i+1;
else cout<<"Nu s-a gasit elementul";
```

Secvența de instrucțiuni pentru căutarea elementului într-o matrice este:

```
int i=0,j=0,n,m,x,a[10][10];
cout<<"n= "; cin>>n;
cout<<"m= "; cin>>m;
cout<<"x= "; cin>>x;
..... // se creează matricea
// se parcurge matricea linie cu linie, iar pe fiecare linie
// se parcurg coloanele până se găsește elementul sau
// până se termină de parcurs toate liniile matricei
i=0; j=0;
while (i<n && a[i][j]!=x)
    if (j==n-1){j=0; i++;} else j++;
if (i!=n)
    cout<<"S-a gasit elementul in linia "<<i+1<<"", coloana "<<j+1;
else
    cout<<"Nu s-a gasit elementul";
```

Algoritmi pentru căutarea într-un vector sortat

Unul dintre algoritmi cei mai folosiți în acest caz este **algoritmul de căutare binară**. Acest algoritm are la bază principiul înjumătățirii repetate a domeniului în care se caută elementul, prin împărțirea vectorului a în doi subvectori. Notăm cu st primul indice al vectorului (indicele elementului din stânga) și cu dr ultimul indice al vectorului (indicele elementului din dreapta), iar $mijl$ este indicele elementului din mijlocul vectorului: $mijl=(st+dr)/2$. Se compară valoarea căutată cu valoarea elementului din mijloc. Dacă cele două valori sunt egale, înseamnă că s-a găsit elementul. Dacă nu sunt egale, vectorul a va fi împărțit în doi subvectori:

- ✓ subvectorul din stânga: $a[st]$, $a[st+1]$, ..., $a[mijl-1]$;
- ✓ subvectorul din dreapta: $a[mijl+1]$, ..., $a[dr-1]$, $a[dr]$.

Așadar, operația de căutare constă în identificarea subvectorului în care se poate găsi elementul, prin compararea valorii căutate cu cea a elementului din mijloc, după care se divizează acest subvector în doi subvectori ș.a.m.d. până când se găsește elementul, sau până când nu se mai poate face împărțirea în subvectori, ceea ce înseamnă că nu s-a găsit elementul.

Se folosesc următoarele **variabile de memorie**:

- ✓ variabila a pentru vector și variabila n pentru lungimea logică a vectorului;
- ✓ variabila x pentru valoarea care trebuie căutată;
- ✓ variabila i pentru parcurgerea vectorului la citire și afișare;
- ✓ variabila st pentru indicele elementului din stânga al vectorului în care se caută;
- ✓ variabila dr pentru indicele elementului din dreapta al vectorului în care se caută;
- ✓ variabila $mijl$ pentru indicele elementului din mijloc al vectorului care se divizează;

- ✓ variabila *gasit* pentru a ști când s-a găsit elementul: inițial, are valoarea 0 (valoarea logică *False* – nu s-a găsit încă elementul), iar atunci când se găsește elementul, i se atribuie valoarea 1 (valoarea logică *True* – s-a găsit elementul).

Pașii algoritmului, pentru un vector sortat crescător, sunt:

- Pas 1.** Se inițializează indicii *st* și *dr*: *st*=0 și *dr*=*n*-1.
Pas 2. Dacă vectorul poate fi împărțit în subvectori, adică *st*≤*dr*, se merge la **Pas 3**; altfel, se merge la **Pas 6**.
Pas 3. Se calculează indicele elementului din mijlocul vectorului *mijl*=(*st*+*dr*)/2.
Pas 4. Dacă elementul din mijlocul vectorului are valoarea *x* (*a*[*mijl*]==*x*), se merge la **Pas 6**.
Pas 5. Dacă elementul din mijlocul vectorului are valoarea mai mare decât *x* (*a*[*mijl*] > *x*), atunci căutarea se va continua în subvectorul din stânga și se modifică valoarea pentru ultimul indice al vectorului *dr* = *mijl*-1; altfel, căutarea se va continua în subvectorul din dreapta, și se modifică valoarea pentru primul indice al vectorului *st*= *mijl*+1. Se revine la **Pas 2**.
Pas 6. Dacă *st*>*dr*, elementul nu s-a găsit în vector; altfel, elementul s-a găsit în poziția *mijl*+1.

Deoarece căutarea va continua atât timp cât nu s-a găsit elementul și vectorul poate fi împărțit în subvectori, se va folosi o variabilă de memorie *gasit* pentru a se verifica dacă elementul s-a găsit sau nu. Inițial *gasit* are valoarea 0 (corespunde valorii logice *False* – nu s-a găsit elementul). În momentul în care se va găsi elementul, *gasit* va lua valoarea 1 (corespunde valorii logice *True* – s-a găsit elementul).

Secvența de instrucțiuni pentru căutarea binară într-un vector este:

```
int i,n,x,st,dr,mijl,gasit,a[10];
cout<<"n= "; cin>>n;
cout<<"x= "; cin>>x;
..... // se creează vectorul
st=0; dr=n-1; gasit=0;
while (st<=dr && !gasit)
    // cât timp vectorul poate fi împărțit în subvectori
    // și nu s-a găsit încă elementul
    {mijl=(st+dr)/2;
    if (a[mijl]==x) //dacă s-a găsit elementul
        gasit=1;
    else
        //dacă nu s-a găsit elementul se stabilește subvectorul
        //în care se continuă căutarea
        if (x<a[mijl]) dr=mijl-1;
        else st=mijl+1;}
if (st>dr) cout<<"Nu s-a găsit elementul";
else cout<<"S-a găsit elementul în poziția "<<mijl+1;
```

5.4.3. Algoritm pentru ștergerea unui element dintr-un vector

Algoritmul pentru ștergerea dintr-un vector a elementului cu indicele *k* înseamnă deplasarea elementelor vectorului, începând cu indicele *k*+1, cu o poziție spre stânga. Lungimea logică a vectorului se va micșora cu un element și va fi *n*-1.

```
int i,n;
cout<<"n=";
cin>>n;
for (i=0; i<n; i++)
    // se a
    for (i=0; i<n; i++)
```

5.4.4. A

Algoritmul
 nă deplas
 dreapta ș
 se va mă
 lungimea

```
const i
int i,n;
cout<<"n=";
cin>>n;
for (i=0; i<n; i++)
    // se a
    for (i=0; i<n; i++)
        if (n+1)
            {for
                a[k]
                for
                else
                {for
                    a[k]
                    for
```

Stu

Scop: ex
 ment într
 într-un ve

Enunțul
 numere în
 la tastatur

Se va folo
 cu valoare

Exemplu
 #include
 void ma
 {int i,n;
 cout<<"n=";
 cin>>n;
 for (i=0; i<n; i++)

```

int i,n,k,a[10];
cout<<"n= "; cin>>n;
cout<<"k= "; cin>>k;
..... // se creează vectorul
for (i=k;i<n-1;i++) a[i]= a[i+1]; // se șterge elementul
// se afișează vectorul după ștergerea elementului
for (i=0;i<n-1;i++) cout<<a[i]<<" ";

```

5.4.4. Algoritm pentru inserarea unui element într-un vector

Algoritmul pentru inserarea într-un vector a unui element în poziția indicelui k înseamnă deplasarea elementelor vectorului, începând cu indicele $k+1$, cu o poziție spre dreapta și atribuirea noii valori elementului cu indicele k . Lungimea logică a vectorului se va mări cu un element și va fi $n+1$, cu condiția ca $n+1$ să fie mai mic sau egal cu lungimea fizică a vectorului, altfel ultimul element al vectorului se pierde.

```

const int DIM=10
int i,n,k,x,a[DIM];
cout<<"n= "; cin>>n;
cout<<"k= "; cin>>k;
cout<<"x= "; cin>>x; // x conține valoarea care se inserează
..... // se creează vectorul
if (n+1<=DIM)
{for (i=n;i>k;i--) a[i]= a[i-1];
  a[k]= x;
  for (i=0;i<n+1;i++) cout<<a[i]<<" ";}
else
{for (i=n-1;i>k;i--) a[i]= a[i-1];
  a[k]= x;
  for (i=0;i<n;i++) cout<<a[i]<<" ";}

```

Studiu de caz

Scop: exemplificarea modului în care se aplică algoritmi pentru căutarea unui element într-un vector, pentru ștergerea unui element și pentru inserarea unui element într-un vector.

Enunțul problemei 1: Se citesc într-un vector, de la tastatură, cel mult 50 de numere întregi. Să se ștergă primul element care are valoarea x (x se citește de la tastatură).

Se va folosi algoritmul de căutare secvențială pentru a găsi poziția primului element cu valoarea x , și apoi algoritmul de ștergere a elementului din acea poziție.

Exemplul 1

```

#include <iostream.h>
void main()
{int i,n,x,k,a[50];
  cout<<"n= "; cin>>n;
  cout<<"x= "; cin>>x;
  for (i=0;i<n;i++) {cout<<"a["<<i+1<<"]= "; cin>>a[i];}
  i=0;

```



```

while (i<n && a[i]!=x) i++;
if (i!=n)
{
    k=i; //k este poziția elementului care se va șterge
    for (i=k; i<n-1; i++) a[i]=a[i+1];
    for (i=0; i<n-1; i++) cout<<a[i]<<" ";
}
else cout<<"Nu s-a gasit elementul";
}

```

Enunțul problemei 2: Se citesc într-un vector, de la tastatură, cel mult 50 de numere întregi. Să se insereze elementul cu valoarea y înainte de primul element care are valoarea x (x și y se citesc de la tastatură).

Se va folosi algoritmul de căutare secvențială pentru a găsi poziția primului element cu valoarea x , și apoi algoritmul de inserare a valorii y în acea poziție.

Exemplul 2

```

#include <iostream.h>
void main()
{
    const int DIM=50;
    int i,n,k,x,y,a[DIM];
    cout<<"n= "; cin>>n;
    cout<<"x= "; cin>>x;
    cout<<"y= "; cin>>y;
    for (i=0; i<n; i++) {cout<<"a["<<i+1<<"]= "; cin>>a[i];}
    for (i=0; i<n && a[i]!=x; i++);
    if (i!=n)
    {
        k=i;
        if (n+1<=DIM)
        {
            for (i=n; i>k; i--) a[i]=a[i-1];
            a[k]=y;
            for (i=0; i<n+1; i++) cout<<a[i]<<" ";
        }
        else
        {
            for (i=n-1; i>k; i--) a[i]=a[i-1];
            a[k]=y;
            for (i=0; i<n; i++) cout<<a[i]<<" ";
        }
    }
    else cout<<"Nu s-a gasit elementul";
}

```

Enunțul problemei 3: Se citesc într-un vector, de la tastatură, cel mult 50 de numere întregi. Vectorul este sortat crescător. Să se insereze un element cu valoarea x , astfel încât să se păstreze ordonarea crescătoare a elementelor vectorului (x se citește de la tastatură).

Se va folosi algoritmul de căutare binară într-un vector sortat, pentru a găsi poziția în care trebuie inserat x , și apoi algoritmul de inserare a elementului cu valoarea x în această poziție. Pentru a crește eficiența algoritmului se vor trata separat cazurile în care elementul va fi inserat pe prima poziție sau după ultima poziție (nu se mai execută secvența de căutare).

Exemplul 3

```

#include <iostream.h>
void main()

```

```

{const i
int i,k
cout<<"
cout<<"
for (i=
if (x<
{for
else
if (x<
else
{
wh
}
}

```

5.4.5. Alg

Acești algoritmi
există o relație

Aranjarea ele

1. Într-un al

✓ algorit

✓ algorit

2. În același

✓ algorit

✓ algorit

✓ algorit

✓ algorit

Algoritmii de so

ui. Pentru ord

pentru compara

dul în care se e

care reprezintă

vectorul este or

Algoritmul de

în această meto

```

{const int DIM=50;
int i,k,n,x,s,d,m,a[DIM];
cout<<"n="; cin>>n;
cout<<"x="; cin>>x;
for (i=0;i<n;i++) {cout<<"a["<<i+1<<"]= "; cin>>a[i];}
if (x<=a[0])
    {for (i=n;i>=1;i--) a[i]=a[i-1]; a[0]=x;}
else
    if (x>=a[n-1]) a[n]=x;
    else
    {s=0; d=n-1;
    while (s<d)
    {m=(s+d)/2;
    if (a[m]>=x) s=m+1;
    else d=m-1;}
    k=d+1; //sau k=s;
    if (n+1<=DIM)
    {for (i=n;i>k;i--) a[i]=a[i-1]; a[k]=x;
    for (i=0;i<n+1;i++) cout<<a[i]<<" ";}
    else
    {for (i=n-1;i>k;i--) a[i]=a[i-1]; a[k]=x;
    for (i=0;i<n;i++) cout<<a[i]<<" ";}
    }
}

```



5.4.5. Algoritmi pentru sortarea unui vector

Acești algoritmi rearanjează elementele vectorului astfel încât între valorile lor să existe o relație de ordine (ordonare crescătoare sau ordonare descrescătoare).

Aranjarea elementelor se poate face:

1. **Într-un alt vector.** În acest caz puteți folosi următorii algoritmi:
 - ✓ algoritmul de sortare prin **metoda inserării**,
 - ✓ algoritmul de sortare prin **metoda numărării**.
2. **În același vector.** În acest caz puteți folosi următorii algoritmi:
 - ✓ algoritmul de sortare prin **metoda selecției directe**,
 - ✓ algoritmul de sortare prin **metoda bulelor**,
 - ✓ algoritmul de sortare prin **metoda inserării directe**,
 - ✓ algoritmul de sortare prin **metoda inserării rapide**.

Algoritmii de sortare prezentați se pot folosi pentru ordonarea crescătoare a vectorului. Pentru ordonarea descrescătoare, este suficient să negați expresiile folosite pentru comparare. Pentru fiecare dintre algoritmii de sortare prezentați, urmăriți modul în care se execută operația de sortare pe vectorul cu 4 elemente $a = \{4, 3, 2, 1\}$ care reprezintă cazul cel mai dezavantajos pentru un vector cu patru elemente (vectorul este ordonat invers).

Algoritmul de sortare prin metoda inserării

În această metodă se folosesc doi vectori:

- ✓ vectorul sursă (vectorul nesortat): a ;
- ✓ vectorul destinație (vectorul sortat): b .

Elementele vectorului sursă $a[i]$ se copiază în vectorul destinație prin inserare în poziția corespunzătoare, astfel încât în vectorul destinație să fie respectată relația de ordine.

Se folosesc următoarele **variabile de memorie**:

- ✓ variabilele a și b pentru vectori și variabila n pentru lungimea logică a vectorilor; primul element din vectorul b ($b[0]$) este inițializat cu valoarea primului element din vectorul a ($a[0]$).
- ✓ variabila i pentru parcurgerea vectorului sursă în vederea preluării elementului $a[i]$ pentru a fi inserat în vectorul b : inițial (la selectarea primului element – $a[1]$) ea are valoarea 1, la fiecare nouă selectare a unui element $a[i]$ se incrementează cu 1, iar operația de parcurgere se termină atunci când variabila i are valoarea $n-1$ (a fost selectat ultimul element – $a[n-1]$);
- ✓ variabila j la parcurgerea vectorului destinație b de la stânga la dreapta, pentru a găsi poziția în care trebuie inserat elementul $a[i]$ conform relației de ordine; la fiecare operație de căutare a poziției în care trebuie inserat noul element $a[i]$, ea va fi inițializată cu indicele primului element din vectorul destinație (0), și se incrementează cu 1 până se găsește poziția de inserare sau până se ajunge la sfârșitul vectorului b (elementul cu indicele $i-1$);
- ✓ variabila k la parcurgerea vectorului destinație b de la dreapta la stânga, pentru a deplasa elementele cu o poziție spre dreapta în vederea eliberării poziției în care se va insera $a[i]$: la fiecare nouă operație de inserare a unui element $a[i]$, ea va fi inițializată cu indicele ultimului element din vectorul destinație (i), și se decrementează cu 1 pentru fiecare deplasare a unui element, până când este deplasat și elementul din poziția de inserare j ;

Pașii algoritmului sunt:

- Pas 1.** Se copiază elementul $a[0]$ în vectorul b pe prima poziție: $b[0]=a[0]$.
- Pas 2.** Se inițializează variabila i cu 1 pentru a începe procesul de copiere din vectorul a în vectorul b .
- Pas 3.** Se inițializează variabila j cu valoarea 0 pentru a începe procesul de căutare a poziției în care va fi inserat elementul $a[i]$.
- Pas 4.** Se compară elementul $a[i]$ cu elementul $b[j]$. Dacă $a[i]>b[j]$, înseamnă că nu s-a găsit încă poziția și se trece la pasul următor; altfel, s-a găsit poziția și trece la **Pas 7**.
- Pas 5.** Se incrementează valoarea lui j cu 1: $j=j+1$ pentru a continua procesul de căutare a poziției de inserare.
- Pas 6.** Se compară j cu numărul de elemente din vectorul destinație ($i-1$). Dacă $j \leq i-1$ se revine la pasul **Pas 4**; altfel, s-a găsit poziția și trece la **Pas 7**.
- Pas 7.** Se inițializează variabila k cu valoarea i pentru a începe procesul de deplasare a elementelor din vectorul b spre dreapta: $k=i$.
- Pas 8.** Se deplasează un element spre dreapta: $b[i]=b[i-1]$.
- Pas 9.** Se decrementează valoarea lui k cu 1: $k=k-1$.
- Pas 10.** Se compară k cu ultima poziție în care se deplasează un element (poziția imediat următoare poziției de inserare j). Când $k \geq j+1$ se revine la **Pas 8**;

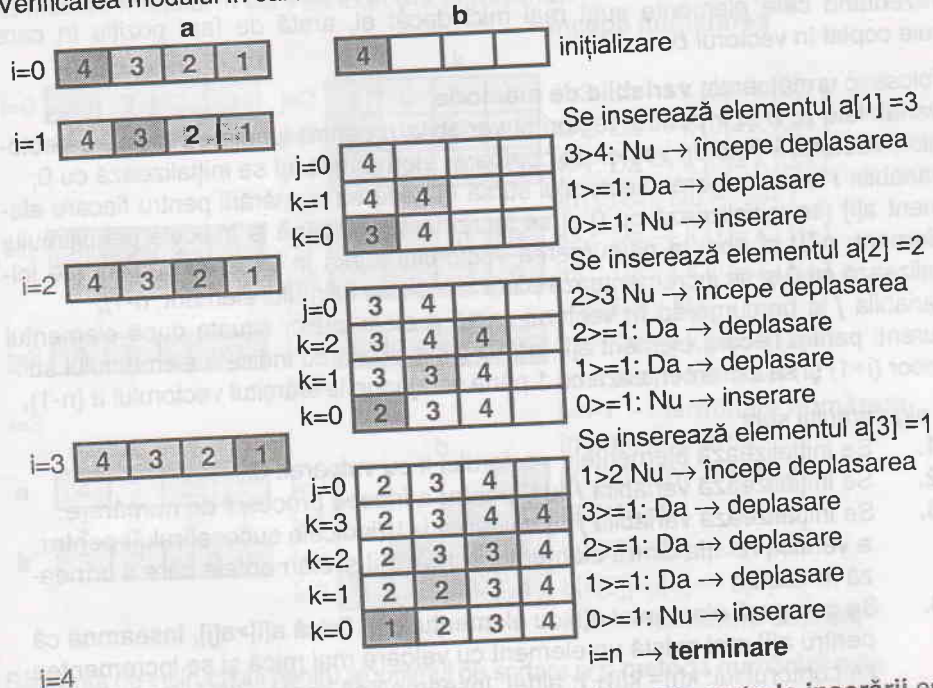
```
int i
cout<
for (
b[0]=
for (
{
w
f
b
}
for (
```

Informatică

altfel, se trece la pasul următor.

- Pas 11.** Se inserează elementul $a[i]$ în vectorul b poziția j : $b[j]=a[i]$.
Pas 12. Se incrementează valoarea lui i cu 1: $i=i+1$, pentru a continua procesul de copiere a unui element din vectorul a în vectorul b .
Pas 13. Se compară variabila i cu indicele ultimului element din vectorul sursă. Dacă $i < n-1$ se revine la **Pas 3**; altfel, se trece la pasul următor.
Pas 14. Vectorul este sortat crescător.

Verificarea modului în care se execută algoritmul:



Secvența de instrucțiuni pentru algoritmul de sortare prin metoda inserării este:

```
int i, j, n, k, a[50], b[50];
cout << "n = "; cin >> n;
for (i=0; i<n; i++) {cout << "a[" << i << "] = "; cin >> a[i];}
b[0]=a[0];
for (i=1; i<n; i++)
{
    j=0;
    while (j<=i-1 && a[i]>b[j]) j++;
    for (k=i; k>=j+1; k--) b[k]=b[k-1];
    b[j]=a[i];
}
for (i=0; i<n; i++) cout << b[i] << " ";
```

Algoritmul de sortare prin metoda numărării

În această metodă se folosesc trei vectori:

- ✓ vectorul sursă (vectorul nesortat): a ;

- ✓ vectorul destinație (vectorul sortat): b ;
- ✓ vectorul numărător (vector de contoare): k .

Elementele vectorului sursă $a[i]$ se copiază în vectorul destinație prin inserare în poziția corespunzătoare, astfel încât în vectorul destinație să fie respectată relația de ordine. Pentru a se cunoaște poziția în care va fi inserat fiecare element, se parcurge vectorul sursă a și se numără în vectorul k , pentru fiecare element $a[i]$, câte elemente au valoarea mai mică decât a lui. Fiecare element al vectorului k este un contor: elementul $k[i]$ este un contor pentru elementul $a[i]$. Valoarea contorului $k[i]$ pentru elementul $a[i]$, reprezentând câte elemente sunt mai mici decât el, arată de fapt poziția în care trebuie copiat în vectorul b .

Se folosesc următoarele **variabile de memorie**:

- ✓ variabilele a , b și k pentru vectori și variabila n pentru lungimea logică a vectorilor; vectorul k fiind un vector de contoare, elementele lui se inițializează cu 0;
- ✓ variabila i la parcurgerea vectorului sursă în vederea numărării pentru fiecare element $a[i]$ (se inițializează cu 0 și se incrementează până la indicele penultimului element: $n-2$) și apoi la parcurgerea vectorului sursă în vederea copierii (se inițializează cu 0 și se incrementează până la indicele ultimului element: $n-1$);
- ✓ variabila j la parcurgerea în vectorul sursă a elementelor situate după elementul curent: pentru fiecare element $a[i]$, ea va fi inițializată cu indicele elementului succesor ($i+1$) și se incrementează cu 1 până se ajunge la sfârșitul vectorului a ($n-1$).

Pașii algoritmului sunt:

- Pas 1.** Se inițializează elementele vectorului k cu valoarea 0.
- Pas 2.** Se inițializează variabila i cu 0 pentru a începe procesul de numărare.
- Pas 3.** Se inițializează variabila j cu valoarea $i+1$ (indicele succesorului) pentru a verifica relația dintre elementul curent $a[i]$ și elementele care îi urmează în vector.
- Pas 4.** Se compară elementul $a[i]$ cu elementul $a[j]$. Dacă $a[i] > a[j]$, înseamnă că pentru $a[i]$ mai există un element cu valoare mai mică și se incrementează contorul lui: $k[i] = k[i] + 1$; altfel, înseamnă că pentru $a[j]$ mai există un element cu valoare mai mică și se incrementează contorul lui: $k[j] = k[j] + 1$.
- Pas 5.** Se incrementează valoarea lui j cu 1: $j = j + 1$ pentru a continua procesul de comparare.
- Pas 6.** Se compară j cu numărul de elemente din vectorul sursă care trebuie vizitate în vederea comparării. Dacă $j < n$ se revine la pasul **Pas 4**; altfel, se trece la pasul următor.
- Pas 7.** Se incrementează valoarea lui i cu 1: $i = i + 1$, pentru a continua procesul de numărare pentru următorul element din vectorul sursă.
- Pas 8.** Se compară variabila i cu indicele ultimului element din vectorul sursă. Dacă $i < n-1$ se revine la **Pas 3**; altfel, s-a terminat procesul de numărare și se trece la pasul următor.
- Pas 9.** Se inițializează variabila i cu 0 pentru a începe procesul de copiere cu elementul $a[0]$.
- Pas 10.** Se copiază elementul $a[i]$ în vectorul destinație în poziția indicată de contorul $k[i]$: $b[k[i]] = a[i]$.

Infor

Pas 1

Pas 12

Pas 13

Verifica

i=0

i=1

i=2

i=3

a

k

Secvență

```
int i,
cout<<
for (i
for (i
for
```

```
for (i
for (i
```

Algoritru

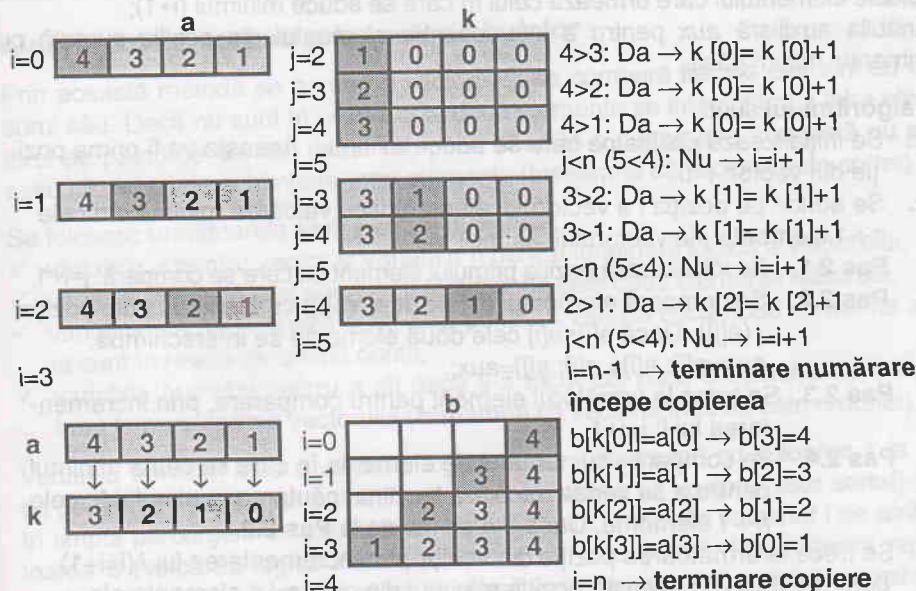
Prin acea
din cele
mai mică
treia elen

Se folose
✓ variab

- Pas 11.** Se incrementează valoarea lui i cu 1: $i=i+1$, pentru a continua procesul de copiere pentru următorul element din vectorul sursă.
- Pas 12.** Se compară variabila i cu indicele ultimului element din vectorul sursă. Dacă $i < n$ se revine la **Pas 10**; altfel, s-a terminat procesul de copiere și se trece la pasul următor.
- Pas 13.** Vectorul b este sortat crescător.

Verificarea modului în care se execută algoritmul:

începe numărarea



Secvența de instrucțiuni pentru algoritmul de sortare prin metoda numărării este:

```
int i, j, n, a[50], b[50], k[50] = {0};
cout << "n = "; cin >> n;
for (i = 0; i < n; i++) {cout << "a[" << i + 1 << "] = "; cin >> a[i];}
for (i = 0; i < n - 1; i++)
    for (j = i + 1; j < n; j++)
        if (a[i] > a[j]) k[i]++;
        else k[j]++;
for (i = 0; i < n; i++) b[k[i]] = a[i];
for (i = 0; i < n; i++) cout << b[i] << " ";
```

Algoritmul de sortare prin metoda selecției directe

Prin această metodă se aduce pe prima poziție elementul cu valoarea cea mai mică din cele n elemente ale vectorului, apoi se aduce pe poziția a doua elementul cu cea mai mică valoare din ultimele $n-1$ elemente ale vectorului, apoi se aduce pe poziția a treia elementul cu cea mai mică valoare din ultimele $n-2$ elemente ale vectorului etc.

Se folosesc următoarele **variabile de memorie**:

- ✓ variabila a pentru vector și variabila n pentru lungimea logică a vectorului,

- ✓ variabila i la parcurgerea vectorului pentru a aduce pe poziția i minimul dintre ultimele elemente (ultimele $n-i-1$ elemente): se inițializează cu 0 (se aduce pe prima poziție valoarea minimă din vector), se incrementează cu 1 (se aduce pe următoarea poziție valoarea minimă din restul elementelor), iar operația de parcurgere se termină atunci când variabila i are valoarea $n-2$ (s-a adus pe penultima poziție minimul din elementul penultim și elementul ultim al vectorului);
- ✓ variabila j la parcurgerea ultimelor $n-i-1$ elemente pentru a găsi valoarea minimă; la fiecare nouă operație de parcurgere a ultimelor elemente, ea va fi inițializată cu indicele elementului care urmează celui în care se aduce minimul ($i+1$);
- ✓ variabila auxiliară aux pentru a interschimba elementul din poziția curentă cu elementul minim găsit.

Pașii algoritmului sunt:

- Pas 1.** Se inițializează poziția pe care se aduce minimul. Aceasta va fi prima poziție din vector: $i=0$.
- Pas 2.** Se aduce pe poziția i a vectorului elementul cu valoarea minimă din cele $n-i-1$ elemente ale vectorului, astfel:
- Pas 2.1.** Se inițializează poziția primului element cu care se compară: $j=i+1$.
- Pas 2.2.** Se compară elementul din poziția i ($a[i]$) cu elementul din poziția j ($a[j]$). Dacă $a[j] < a[i]$ cele două elemente se interschimbă:
 $aux=a[i]$; $a[i]=a[j]$; $a[j]=aux$;
- Pas 2.3.** Se trece la următorul element pentru comparare, prin incrementarea lui j : $j=j+1$.
- Pas 2.4.** Se compară j cu numărul de elemente în care se caută minimul pentru a se vedea dacă s-a terminat căutarea minimului în cele $n-i-1$ elemente. Dacă $j < n$ se revine la **Pas 2.2**.
- Pas 3.** Se trece la următoarea poziție din vector prin incrementarea lui i ($i=i+1$) pentru a duce pe această poziție minimul din cele $n-i-1$ elemente ale vectorului.
- Pas 4.** Se compară i cu numărul de poziții ale vectorului în care trebuie aduse valorile minime. Dacă $i < n-1$ se revine la **Pas 2**.
- Pas 5.** Vectorul este sortat crescător.

Verificarea modului în care se **execută algoritmul**:

$i=0$

4	3	2	1
---	---	---	---

$j=1$

4	3	2	1
---	---	---	---

 interschimbare
 $j=2$

3	4	2	1
---	---	---	---

 interschimbare
 $j=3$

2	4	3	1
---	---	---	---

 interschimbare

$i=1$

1	4	3	2
---	---	---	---

$j=2$

1	4	3	2
---	---	---	---

 interschimbare
 $j=3$

1	3	4	2
---	---	---	---

 interschimbare

$i=2$

1	2	4	3
---	---	---	---

$j=3$

1	2	4	3
---	---	---	---

 interschimbare
 $i=n-1 \rightarrow$ **terminare**

$i=3$

1	2	3	4
---	---	---	---

Secvența de instrucțiuni pentru algoritmul de sortare prin metoda selecției directe este:

```
int i, j, n, aux, a[50];
cout<<"n = "; cin>>n;
for (i=0; i<n; i++) {cout<<"a["<<i+1<<"]= "; cin>>a[i];}
for (i=0; i<n-1; i++)
    for (j=i+1; j<n; j++)
        if (a[j]<a[i]) {aux=a[i]; a[i]=a[j]; a[j]=aux;}
for (i=0; i<n; i++) cout<<a[i]<<" ";
```

Algoritmul de sortare prin metoda bulelor

Prin această metodă se parcurge vectorul și se compară fiecare element cu succesorul său. Dacă nu sunt în ordine cele două elemente se interschimbă între ele. Vectorul se parcurge de mai multe ori, până când la o parcurgere completă nu se mai execută nici o interschimbare între elemente (înseamnă că vectorul este sortat).

Se folosesc următoarele **variabile de memorie**:

- ✓ variabila *a* pentru vector și variabila *n* pentru lungimea logică a vectorului,
- ✓ variabila auxiliară *aux* pentru a interschimba cele două elemente alăturate,
- ✓ variabila *i* la parcurgerea vectorului pentru a verifica dacă două elemente alăturate sunt în relația de ordine dorită,
- ✓ variabila *terminat* pentru a ști dacă s-a făcut cel puțin o operație de interschimbare la parcurgerea vectorului (înseamnă că vectorul încă nu este ordonat).

Variabila *terminat* este o variabilă logică ce se inițializează cu valoarea 1 la începutul parcurgerii (valoarea logică *True* – se presupune că vectorul este sortat) și, dacă în timpul parcurgerii se execută o interschimbare, variabilei *terminat* i se atribuie valoarea 0 (valoarea logică *False* – vectorul nu este sortat). Parcurgerea repetată a vectorului se termină atunci când, la sfârșitul unei parcurgeri, variabila *terminat* nu își modifică valoarea (își păstrează valoarea 1).

Pașii algoritmului sunt:

- Pas 1.** Se inițializează variabila *terminat* cu valoarea 1.
- Pas 2.** Se inițializează variabila *i* cu 0 ($i=0$) corespunzătoare primului element din vector.
- Pas 3.** Se compară elementul din poziția *i* ($a[i]$) cu succesorul său ($a[i+1]$). Dacă $a[i+1]<a[i]$ cele două elemente se interschimbă ($aux=a[i]$; $a[i]=a[i+1]$; $a[i+1]=aux$;) și variabilei *terminat* i se atribuie valoarea 0.
- Pas 4.** Se trece la următoarea poziție din vector prin incrementarea lui *i* ($i=i+1$).
- Pas 5.** Se compară *i* cu numărul de poziții ale vectorului care trebuie parcurs. Dacă $i<n-1$ se revine la **Pas 3**; altfel, se trece la pasul următor.
- Pas 6.** Se verifică valoarea variabilei *terminat*. Dacă *terminat*=0 se revine la **Pas 1**.
- Pas 7.** Vectorul este sortat crescător.

Verificarea modului în care se execută algoritmul:

prima parcurgere
terminat=1

i=0

4	3	2	1
---	---	---	---

interschimbare; **terminat=0**

i=1

3	4	2	1
---	---	---	---

interschimbare; **terminat=0**

i=2

3	2	4	1
---	---	---	---

interschimbare; **terminat=0**

a doua parcurgere
terminat=1

i=0

3	2	1	4
---	---	---	---

interschimbare; **terminat=0**

i=1

2	3	1	4
---	---	---	---

interschimbare; **terminat=0**

i=2

2	1	3	4
---	---	---	---

a treia parcurgere
terminat=1

i=0

2	1	3	4
---	---	---	---

interschimbare; **terminat=0**

i=1

1	2	3	4
---	---	---	---

terminat=0

i=2

1	2	3	4
---	---	---	---

a patra parcurgere
terminat=1

i=0

1	2	3	4
---	---	---	---

i=1

1	2	3	4
---	---	---	---

i=2

1	2	3	4
---	---	---	---

terminat=1 → terminare

Secvența de instrucțiuni pentru algoritmul de sortare prin metoda bulelor este:

```
int i,n,terminat=0,aux,a[50];
cout<<"n="; cin>>n;
for (i=0;i<n;i++) {cout<<"a["<<i+1<<"]= "; cin>>a[i];}
while (!terminat)
{
    terminat=1;
    for (i=0;i<n-1;i++)
        if (a[i]>a[i+1])
            {aux=a[i]; a[i]=a[i+1]; a[i+1]=aux; terminat=0;}
}
for (i=0;i<n;i++) cout<<a[i]<<" ";
```

Algoritmul de sortare prin metoda inserării directe

Prin această metodă se împarte vectorul în doi subvectori:

- ✓ subvectorul sursă: $a[i], a[i+1], \dots, a[n-1]$;
- ✓ subvectorul destinație: $a[0], a[1], \dots, a[i-1]$.

Elementul $a[i]$ din subvectorul sursă este inserat în subvectorul destinație conform relației de ordine. Din această cauză vectorul destinație va fi tot timpul un vector ordonat.

Se folosesc următoarele **variabile de memorie**:

- ✓ variabila a pentru vector și variabila n pentru lungimea fizică a vectorului;
- ✓ variabila i la împărțirea vectorului în subvectori: inițial (la prima împărțire în subvectori) ea are valoarea 1 (subvectorul destinație are un singur element), la fiecare nouă împărțire în subvectori se incrementează cu 1, iar operația de împărțire în subvectori se termină atunci când variabila i are valoarea $n-1$ (vectorul sursă nu mai conține nici un element);
- ✓ variabila j la parcurgerea secvenței destinație de la dreapta la stânga, pentru a găsi poziția în care trebuie inserat elementul $a[i]$ conform relației de ordine; la fiecare nouă operație de împărțire în subvectori, ea va fi inițializată cu indicele ultimului element din vectorul destinație ($i-1$);
- ✓ variabila auxiliară aux , pentru a salva elementul $a[i]$, care se pierde în urma deplasării spre dreapta în vectorul destinație a elementelor care urmează după poziția în care va fi inserat.

Pașii algoritmului sunt:

- Pas 1.** Se inițializează variabila i cu valoarea 1 (se face prima împărțire în subvectori).
- Pas 2.** Se salvează elementul $a[i]$ în variabila aux : $aux=a[i]$.
- Pas 3.** Se inițializează variabila j cu valoarea $i-1$ (indicele ultimului element din vectorul destinație).
- Pas 4.** Se compară variabila aux cu elementul din poziția j ($a[j]$) din vectorul destinație. Dacă $aux < a[j]$, se deplasează elementul $a[j]$ spre dreapta: $a[j+1]=a[j]$; altfel, se trece la **Pas 7**.
- Pas 5.** Se decrementează valoarea lui j cu 1: $j=j-1$.
- Pas 6.** Se compară j cu indicele primului element din vectorul destinație. Dacă $j > 0$ se revine la **Pas 4**; altfel, se trece la pasul următor.
- Pas 7.** Se compară variabila aux cu elementul din poziția j ($a[j]$) din vectorul destinație. Dacă $aux >= a[j]$ se inserează elementul aux în poziția $j+1$: $a[j+1]=aux$; altfel, se deplasează elementul din poziția 0 în poziția 1 ($a[1]=a[0]$) și se inserează elementul aux în poziția 0: $a[0]=aux$.
- Pas 8.** Se face următoarea împărțire în subvectori prin incrementarea lui i ($i=i+1$).
- Pas 9.** Se compară i cu indicele ultimului element din vectorul sursă. Dacă $i < n-1$ se revine la **Pas 2**; altfel, se trece la pasul următor.
- Pas 10.** Vectorul este sortat crescător.

Verificarea modului în care se execută algoritmul:

i=1	<table><tr><td>4</td><td>3</td><td>2</td><td>1</td></tr></table>	4	3	2	1	j=0	<table><tr><td>4</td><td>4</td><td>2</td><td>1</td></tr><tr><td>3</td><td>4</td><td>2</td><td>1</td></tr></table>	4	4	2	1	3	4	2	1	aux=3 deplasare spre dreapta a[1]=aux								
4	3	2	1																					
4	4	2	1																					
3	4	2	1																					
i=2	<table><tr><td>3</td><td>4</td><td>2</td><td>1</td></tr></table>	3	4	2	1	j=1	<table><tr><td>3</td><td>4</td><td>4</td><td>1</td></tr><tr><td>3</td><td>3</td><td>4</td><td>1</td></tr></table>	3	4	4	1	3	3	4	1	aux=2 deplasare spre dreapta deplasare spre dreapta a[1]=aux								
3	4	2	1																					
3	4	4	1																					
3	3	4	1																					
i=3	<table><tr><td>2</td><td>3</td><td>4</td><td>1</td></tr></table>	2	3	4	1	j=0	<table><tr><td>2</td><td>3</td><td>4</td><td>1</td></tr><tr><td>2</td><td>3</td><td>4</td><td>1</td></tr></table>	2	3	4	1	2	3	4	1	aux=1 deplasare spre dreapta deplasare spre dreapta deplasare spre dreapta a[1]=aux								
2	3	4	1																					
2	3	4	1																					
2	3	4	1																					
i=4	<table><tr><td>1</td><td>2</td><td>3</td><td>4</td></tr></table>	1	2	3	4	j=2	<table><tr><td>2</td><td>3</td><td>4</td><td>4</td></tr><tr><td>2</td><td>3</td><td>3</td><td>4</td></tr><tr><td>2</td><td>2</td><td>3</td><td>4</td></tr><tr><td>1</td><td>2</td><td>3</td><td>4</td></tr></table>	2	3	4	4	2	3	3	4	2	2	3	4	1	2	3	4	i=n → terminare
1	2	3	4																					
2	3	4	4																					
2	3	3	4																					
2	2	3	4																					
1	2	3	4																					

Secvența de instrucțiuni pentru algoritmul de sortare prin metoda inserării directe este:

```
int i,j,n,aux,a[50];
cout<<"n="; cin>>n;
for (i=0;i<n;i++) {cout<<"a["<<i+1<<"]= "; cin>>a[i];}
for (i=1;i<n;i++)
{aux=a[i]; j=i-1;
while (j>0 && aux<a[j]) {a[j+1]=a[j]; j--;}
if (aux>=a[j]) a[j+1]=aux;
else {a[1]=a[0]; a[0]=aux;}
}
for (i=0;i<n;i++) cout<<a[i]<<" ";
```


Algoritmul de sortare prin metoda inserării rapide

Acest algoritm folosește aceeași împărțire a vectorului în doi subvectori (sursă și destinație), la fel ca și metoda inserării directe. Deoarece vectorul destinație este un vector ordonat, căutarea poziției în care va fi inserat elementul $a[i]$ se poate face nu secvențial (așa cum se procedează în cazul inserării directe când se folosește variabila j pentru parcurgerea secvențială a vectorului destinație), ci prin algoritmul de căutare binară. Subvectorul destinație este împărțit în doi subvectori, se examinează relația de ordine dintre elementul de la mijlocul subvectorului și elementul $a[i]$ și se stabilește dacă elementul $a[i]$ va trebui inserat în prima jumătate a subvectorului sau în a doua jumătate. Operația de divizare a subvectorului se continuă până se găsește poziția în care urmează să fie inserat $a[i]$.

Secvența de instrucțiuni pentru algoritmul de sortare prin **metoda inserării rapide** este:

```
int i,j,n,aux,st,dr,mijl,a[50];
cout<<"n="; cin>>n;
for (i=0;i<n;i++) {cout<<"a["<<i+1<<"]= "; cin>>a[i];}
for (i=1;i<n;i++)
{aux=a[i]; st=0; dr=i-1;
  while (st<=dr)
  {mijl=(st+dr)/2;
   if (aux<a[mijl]) dr=mijl-1;
   else st=mijl+1;
   j=i-1;
   while (j>=st) {a[j+1]=a[j]; j=j-1;}
   a[st]=aux;
  }
}
for (i=0;i<n;i++) cout<<a[i]<<" ";
```

5.4.6. Algoritm pentru interclasarea a doi vectori

Interclasarea a doi vectori înseamnă reuniunea celor doi vectori urmată de ordonarea elementelor vectorului rezultat.

Pentru interclasarea a doi vectori există **algoritmul pentru interclasarea a doi vectori sortați**.

Presupunem că cei doi vectori sunt sortați (crescător sau descrescător). Prin acest algoritm se parcurg simultan cei doi vectori sursă pentru a se compara un element dintr-un vector cu un element din celălalt vector. Elementul cu valoarea mai mică respectiv mai mare, este copiat în vectorul destinație și șters din vectorul sursă. Procesul continuă până când este epuizat unul dintre vectori. Elementele rămase în celălalt vector se adaugă la sfârșitul vectorului destinație.

Se folosesc următoarele **variabile de memorie** pentru:

- ✓ **vectori:** **a** și **b** – vectorii sursă (se presupune că sunt deja sortați crescător sau descrescător) și **c** – vectorul rezultat (vectorul destinație) care reunește elementele din vectorii **a** și **b**, ordonate după același criteriu ca și cele din vectorii sursă;

- ✓ **lungimile logice ale vectorilor:** n – pentru vectorul a (se introduce de la tastatură) și m – pentru vectorul b (se introduce de la tastatură); lungimea vectorului c se calculează $(n+m)$;
- ✓ **contorii pentru parcurgerea vectorilor:** i – pentru vectorul a , j – pentru vectorul b și k – pentru vectorul c .

Pașii algoritmului (pentu vectori sortați crescător) sunt:

- Pas 1.** Se inițializează variabilele contor, pentru vectorii sursă și pentru vectorul rezultat, cu 0: $i=0$; $j=0$; $k=0$;
- Pas 2.** Se compară elementele $a[i]$ și $b[j]$. Dacă $a[i] < b[j]$ se copiază elementul $a[i]$ în vectorul c prin $c[k]=a[i]$ și se șterge elementul $a[i]$ din vectorul a ; altfel, se copiază elementul $b[j]$ în vectorul c prin $c[k]=b[j]$ și se șterge elementul $b[j]$ din vectorul b . Operația de ștergere nu se execută prin eliminarea efectivă a elementului din vector, ci prin incrementarea contorului: dacă se șterge elementul din vectorul a se incrementează contorul i ($i=i+1$), iar dacă se șterge elementul din vectorul b se incrementează contorul j ($j=j+1$).
- Pas 3.** Se incrementează contorul k pentru că urmează să se copieze un element în acest vector.
- Pas 4.** Se compară contorii vectorilor sursă cu lungimea lor. Dacă fiecare contor este mai mic sau egal cu lungimea vectorului ($i < n$ și $j < m$) se revine la **Pas 2**; altfel, se trece la pasul următor.
- Pas 5.** Se adaugă la sfârșitul vectorului c elementele rămase în celălalt, printr-o simplă operație de copiere: dacă $i < n$ înseamnă că s-a epuizat vectorul b și se vor adăuga la vectorul c elementele rămase în vectorul a (se execută $c[k]=a[i]$; $k=k+1$; $i=i+1$; până când i devine egal cu lungimea n a vectorului a); altfel, s-a epuizat vectorul a și se vor adăuga la vectorul c elementele rămase în vectorul b (se execută $c[k]=b[j]$; $k=k+1$; $j=j+1$; până când j devine egal cu lungimea m a vectorului b).

Secvența de instrucțiuni pentru **interclasarea a doi vectori sortați crescător** este:

```
{int i,j,k,n,m,a[50],b[50],c[100];
cout<<"n="; cin>>n; cout<<"m="; cin>>m;
for (i=0;i<n;i++) {cout<<"a["<<i+1<<"]= "; cin>>a[i];}
for (j=0;j<m;j++) {cout<<"b["<<j+1<<"]= "; cin>>b[j];}
i=0; j=0; k=0;
while (i<n && j<m)
{ if (a[i]<b[j]) {c[k]=a[i]; i++;}
  else {c[k]=b[j]; j++;}
  k++; }
if (i<n)
while (i<n) {c[k]=a[i]; k++; i++;}
else
while (j<m) {c[k]=b[j]; k++; j++;}
for (k=0;k<n+m;k++) cout<<c[k]<<" ";
```


5.4.7. Aplicarea algoritmilor pentru prelucrarea tablourilor de memorie

Folosirea tablourilor de memorie se recomandă atunci când trebuie să prelucrați o colecție de date de același tip și pentru rezolvarea problemei aveți nevoie la un moment dat de toate elementele colecției de date. În continuare vor fi prezentate câteva exemple de probleme de matematică ce pot fi rezolvate cu ajutorul tablourilor de memorie:

Enunțul problemei 1: Să se calculeze produsul scalar a doi vectori **a** și **b**. Dimensiunea vectorilor și coordonatele lor se citesc de la tastatură.

Exemplul 1

```
#include <iostream.h>
void main()
{int i,j,n,p=0,a[50],b[50];
 cout<<"n="; cin>>n;
 for (i=0;i<n;i++) {cout<<"a["<<i+1<<"]= "; cin>>a[i];}
 for (i=0;i<n;i++) {cout<<"b["<<i+1<<"]= "; cin>>b[i];}
 for (i=0;i<n;i++) p=p+a[i]*b[i];
 cout<<"produsul scalar="<<p;
}
```

Enunțul problemei 2: Să se calculeze produsul a două polinoame. Gradul polinoamelor și coeficienții lor se citesc de la tastatură.

Exemplul 1

```
#include <iostream.h>
void main()
{int i,j,n,m,a[50],b[50],c[50]={0};
 cout<<"Primul polinom"<<endl;
 cout<<"Gradul = "; cin>>n;
 cout<<"Coeficientii: "<<endl;
 for (i=0;i<=n;i++) {cout<<"a("<<i<<")= "; cin>>a[i];}
 cout<<"Al doilea polinom"<<endl;
 cout<<"Gradul = "; cin>>m;
 cout<<"Coeficientii: "<<endl;
 for (j=0;j<=m;j++) {cout<<"b("<<j<<")= "; cin>>b[j];}
 for (i=0;i<=n;i++)
   for (j=0;j<=m;j++) c[i+j]=c[i+j]+a[i]*b[j];
 cout<<"Coeficientii polinomului produs:"<<endl;
 for (i=0;i<=n+m;i++) cout<<"c("<<i<<")= "<<c[i]<<endl;
}
```

Enunțul problemei 3: Să se afișeze toate numerele prime mai mici decât un număr **n** ($n > 2$) citit de la tastatură.

Această problemă se poate rezolva și folosind algoritmul cunoscut sub numele de **sita lui Eratostene**. Algoritmul constă în generarea unei mulțimi de numere naturale de la 2 până la **n** (sita). Se extrage primul număr (2) care este un număr prim. Se scot din sită toate numerele care se divid cu el. Se extrage din sită următorul număr, care va fi și el tot prim. Se scot din sită toate numerele care se divid cu el. Procesul continuă până când sita rămâne goală (au fost extrase toate numerele:

unele pentru că erau numere prime, iar altele pentru că erau multipli unui număr prim). Implementarea acestui algoritm nu se poate face decât folosind o structură de date.

Se folosesc următoarele **variabile de memorie**:

- ✓ variabila *a* pentru vectorul în care se generează numerele (sita), *b* pentru vectorul în care se extrag numerele prime și variabila *n* pentru lungimea logică a vectorului *a*;
- ✓ variabila *i* pentru parcurgerea vectorului *a* și variabila *j* pentru indicii elementului curent din vectorul *b*;
- ✓ variabila *terminat* pentru a se verifica dacă sita este goală sau nu; poate avea valoarea 0 (valoarea logică *False* – sita nu este goală) sau valoarea 1 (valoarea logică *True* – sita este goală);

Pentru ca algoritmul să fie mai eficient, se vor genera în vectorul *a* numai numerele impare mai mici decât *n*. Eliminarea multiplului unui număr prim din sită se va face atribuindu-i valoarea 0.

Pașii algoritmului sunt:

- Pas 1.** Se generează în vectorul *a* numerele impare mai mici decât *n*. Primul număr din vectorul *a* este 3, care este un număr prim.
- Pas 2.** Se fac inițializările: *terminat*=0, *b*[0]=2; *j*=0 (s-a scris în vectorul *b* numărul prim 2 care nu există în sită).
- Pas 3.** Se parcurge vectorul *a* și se elimină toți multiplii numărului prim *b*[*j*] atribuindu-le valoarea 0.
- Pas 4.** Se parcurge vectorul *a* până se găsește primul număr prim (primul număr diferit de 0).
- Pas 5.** Dacă mai există un număr prim (indicele *i* este mai mic decât lungimea logică a vectorului $(n-2)/2$), se copiază acest număr în vectorul *b*; altfel, se atribuie variabilei *terminat* valoarea 1.
- Pas 6.** Se analizează valoarea variabilei *terminat*. Dacă are valoarea 0, se revine la **Pas 3**.

Exemplul 3

```
#include <iostream.h>
void main()
{int i,j,n,terminat=0,a[50],b[50];
 cout<<"n = "; cin>>n;
 for (i=0;i<(n-1)/2;i++) a[i]=2*i+3;
 b[0]=2; j=0;
 while (!terminat)
 {for (i=0;i<(n-1)/2; i++)
  if ( a[i]%b[j]==0) a[i]=0;
  for (i=0;a[i]==0;i++);
  if (i<(n-1)/2) {j++; b[j]=a[i];}
  else terminat=1;
 }
 for (i=0;i<=j;i++) cout<<b[i]<<" ";
 }
```


Enunțul problemei 4: Se consideră două mulțimi de numere întregi A și B . Să se calculeze:

- a) reuniunea celor două mulțimi: $X = A \cup B$
- b) intersecția celor două mulțimi: $Y = A \cap B$
- c) diferența celor două mulțimi: $Z = A - B$

Exemplul 4_a

```
#include <iostream.h>
void main()
{int i,j,k,n,m,gasit,a[50],b[50],c[100];
cout<<"n="; cin>>n; cout<<"m="; cin>>m;
for (i=0;i<n;i++) {cout<<"a["<<i+1<<"]= "; cin>>a[i];}
for (j=0;j<m;j++) {cout<<"b["<<j+1<<"]= "; cin>>b[j];}
k=0;
for (i=0;i<n;i++) {c[k]=a[i]; k++;}
for (j=0;j<m;j++)
{gasit=0;
for (i=0; i<n && !gasit;i++)
if (a[i]==b[j]) gasit=1;
if (!gasit) {c[k]=b[j]; k++;}
}
for (i=0;i<k;i++) cout<<c[i]<<" ";
```

Exemplul 4_b

```
#include <iostream.h>
void main()
{int i,j,k,n,m,a[50],b[50],c[50];
cout<<"n="; cin>>n; cout<<"m="; cin>>m;
for (i=0;i<n;i++) {cout<<"a["<<i+1<<"]= "; cin>>a[i];}
for (j=0;j<m;j++) {cout<<"b["<<j+1<<"]= "; cin>>b[j];}
for (i=0,k=-1;i<n;i++)
for (j=0;j<m;j++)
if (a[i]==b[j]) {k++; c[k]=a[i];}
if (k==n)
cout<<"Intersectia este multimea vida";
else
for (i=0;i<=k;i++) cout<<c[i]<<" ";
```

Exemplul 4_c

```
#include <iostream.h>
void main()
{int i,j,k,n,m,gasit=0,a[50],b[50],c[50];
cout<<"n="; cin>>n; cout<<"m="; cin>>m;
for (i=0;i<n;i++) {cout<<"a["<<i+1<<"]= "; cin>>a[i];}
for (j=0;j<m;j++) {cout<<"b["<<j+1<<"]= "; cin>>b[j];}
for (i=0,k=0;i<n;i++)
{for (j=0; j<m && !gasit;j++)
if (a[i]==b[j]) gasit=1;
if (!gasit) {c[k]=a[i]; k++;}
}
if (k==0)
cout<<"Diferența este multimea vida";
else
for (i=0;i<k;i++) cout<<c[i]<<" ";
```

Evaluare

Răspundeți:

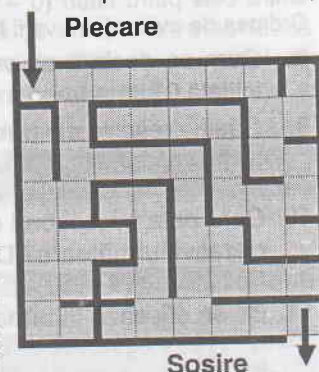
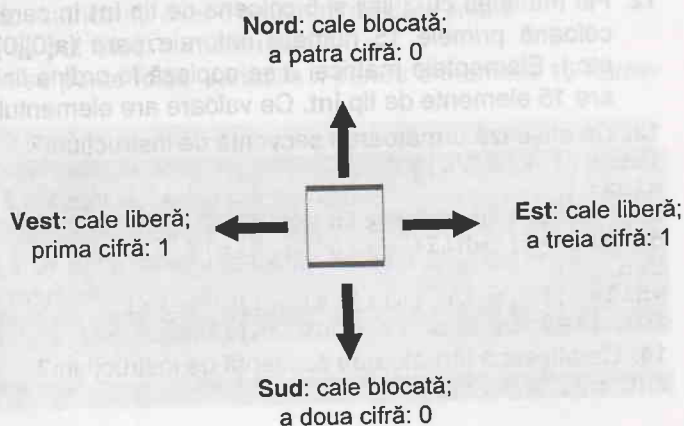
1. Descrieți cum pot fi organizate datele în colecții. Dați exemple de organizare a datelor în colecții.
2. Se consideră o matrice cu 3 linii și 5 coloane. Arătați cum vor fi aranjate în memorie elementele matricei dacă ele sunt înregistrate în ordinea liniilor. Arătați cum vor fi aranjate și în cazul în care ar fi înregistrate în ordinea coloanelor. Ce constatați? Cum vor fi aranjate în memorie elementele acestei matrice dacă va fi implementată în limbajul C++?
3. Construiți o structură de date de tip tabel în care să înregistrați notele elevilor din clasă pe semestrul 1 la disciplina „Informatică”.
4. Dacă elementele unei matrice cu 5 linii și 8 coloane sunt înregistrate în ordinea liniilor începând de la adresa de memorie 100, care este adresa elementului din linia 4 și coloana 5? Datele memorate în matrice sunt de tip `int`.
5. Dacă elementele unei matrice cu m linii și n coloane sunt înregistrate în memoria internă în ordinea coloanelor, care este formula prin care calculați numărul de ordine al elementului din linia i și coloana j ?
6. Se consideră labirintul din figură. Cu ajutorul calculatorului trebuie să se găsească drumul prin labirint.
 - a) Să se găsească metoda de reprezentare a labirintului în memoria internă.
 - b) Să se exprime condiția de ieșire din labirint.

1	2	3	4	5
5	4	3	2	1
1	3	5	2	4

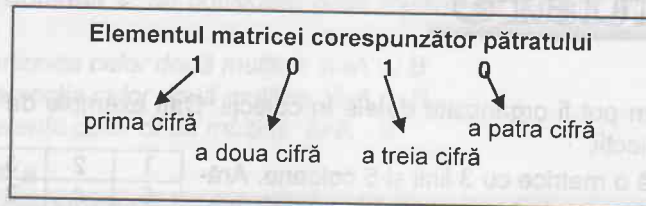
Labirintul reprezintă datele de intrare ale problemei, iar datele de ieșire vor fi reprezentate prin soluțiile sau soluția de parcurgere a labirintului de la **plecare** la **sosire**. Cea mai simplă metodă de reprezentare a labirintului este prin intermediul **taboului bidimensional** (matricea). Fiecare element al matricei va corespunde unui pătrat din labirint și va memora informații despre acest pătrat. Aceste informații se referă la mișcările care pot fi executate în pătrat.

Fiecărui element al matricei i se atribuie un șir de patru cifre binare construit după următoarele reguli:

- ✓ prima cifră: are valoarea 1 dacă se poate executa un pas spre Vest;
- ✓ a doua cifră: are valoarea 1 dacă se poate executa un pas spre Sud;
- ✓ a treia cifră: are valoarea 1 dacă se poate executa un pas spre Est;
- ✓ a patra cifră: are valoarea 1 dacă se poate executa un pas spre Nord.



Pentru pătratul din exemplu:

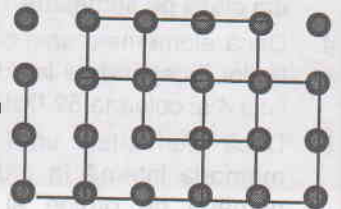


Șirul de patru cifre binare formează numărul binar 1010 care va fi transformat în zecimal. Elementul matricei corespunzător pătratului va memora numărul zecimal întreg 10.

Elementul matricei **L** este **L(i,j)** unde: **i** reprezintă numărul liniei și poate lua valori de la 1 la 7, iar **j** reprezintă numărul coloanei și poate lua valori de la 1 la 8.

Elementul matricei care corespunde pătratului prin care se iese din labirint este: **L(7,8)**. Deci: **sosire = (i == 7) && (j == 8)** și **! sosire = (i != 7) || (j != 8)**

7. Considerați grila din figură formată din puncte și linii. Construiți o matrice de date de intrare care să descrie această grilă.



Fiecare element al matricei va conține informații despre un pătrat din grilă. Așadar, matricea va avea 3 linii și 5 coloane. Pentru fiecare pătrat se va evidenția dacă are sau nu fiecare dintre cele patru laturi (0 – dacă nu are și 1 – dacă are). Ordinea de evidențiere va fi N, E, S, V.

8. Cum se declară un vector cu 10 elemente de tip întreg care să aibă valoarea inițială 0?

9. Găsiți greșelile din următoarele instrucțiuni declarative:

```
const int DIM=10;
int n, a[n], b[dim], B[10], c[10,20];
```

10. Care este succesorul elementului **a[i][j]** al matricei definită prin instrucțiunea declarativă următoare? Dar predecesorul său?

```
int i, j, a[4][5];
```

11. Ce se afișează în urma execuției următoarei secvențe de instrucțiuni?

```
int a[3]={1,2,3}, b[3];
b=a;
cout<<b[0]+b[1]*b[2];
```

12. Fie matricea cu 3 linii și 5 coloane de tip **int** în care se înregistrează coloană cu coloană primele 15 numere naturale pare (**a[0][0]=2; a[1][0]=4; ..., a[0][1]=8; etc.**). Elementele matricei **a** se copiază în ordine linie cu linie, în vectorul **b** care are 15 elemente de tip **int**. Ce valoare are elementul **b[5]**?

13. Ce afișează următoarea secvență de instrucțiuni?

```
int n,i,k,a[20];
cin>>n;
for (i=0;i<n;i++) cin>>a[i];
do cin>>k; while (k<0 || k>n);
i=n;
while (i != k) {a[i]=a[i-1]; i--;}
for (i=0;i<=n;i++) cout>>a[i];
```

14. Ce afișează următoarea secvență de instrucțiuni?

```
int n,i,k,x,a[20];
```

```
cin>>n; cin>>k;
for (i=0;i<n;i++) cin>>a[i];
x=a[k];
for (i=k;i<n-1;i++) a[i]= a[i+1];
a[n-1]=x;
```

15. Ce afișează următoarea secvență de instrucțiuni?

```
int n,i,j,k=0,a[20];
cin>>n;
..... //se citesc elementele vectorului a de la tastatură
for (i=0;i<n-1;i++)
    for (j=i+1;j<n;j++)
        if (a[i]==a[j]) k++;
```

16. Câte comparații se execută pentru sortarea unui vector cu n componente folosind metoda selectării directe?
17. Câte comparații se execută pentru sortarea unui vector cu n componente folosind metoda bulelor?
18. Câte comparații se execută pentru sortarea unui vector cu n componente folosind metoda inserării directe?
19. Câte comparații se execută pentru sortarea unui vector cu n componente folosind metoda inserării rapide?
20. Comparați, din punct de vedere al eficienței, algoritmi de sortare care sortează un vector în el însuși. Folosiți pentru analizarea complexității lor numărul de operații elementare. Considerați ca operație elementară comparația.
21. Comparați, din punct de vedere al eficienței, algoritmi de sortare care sortează un vector în el însuși. Folosiți pentru analizarea complexității lor timpul maxim de execuție. Considerați cazul cel mai defavorabil: un vector sortat în ordine inversă.

Adevărat sau fals:

1. Tabloul de memorie este o structură de date externă.
2. Tabloul de memorie este o structură de date dinamică deoarece i se poate modifica lungimea logică la execuția programului.
3. Tabloul de memorie se creează într-o zonă continuă de memorie internă.
4. Tabloul de memorie este întotdeauna o structură de date temporară.
5. Tabloul de memorie de tip matrice nu este o structură de date liniară.
6. Pentru indicii unui vector se pot folosi numere reale.
7. Cu declarațiile următoare se poate folosi variabila x pentru a memora 15 numere întregi:

```
typedef int vec int[3];_
vec_int x[5];
```

8. Pentru variabila x definită anterior este corectă atribuirea:

```
x[2] = {1,2,3};
```

9. Cu declarațiile următoare se obțin tablourile de memorie x și y , echivalente din punct de vedere al implementării:

```
typedef int vec int[3];_
vec int x[5];
int y[5][3];
```


10. Cu declarația următoare se obțin tablourile de memorie x și y , echivalente din punct de vedere al implementării:

```
int x[3][5], y[5][3];
```

11. Cu declarația următoare se obțin tablourile de memorie x și y , echivalente din punct de vedere al implementării:

```
int x[15], y[5][3];
```

12. În matricea pătrată a , cu n linii și n coloane, elementul $a[n-2][n-2]$ aparține diagonalei principale.

13. În matricea pătrată a , cu n linii și n coloane, elementul $a[n-2][3]$ aparține diagonalei secundare.

14. În matricea pătrată a , cu 10 linii și 10 coloane, elementul $a[3][2]$ se găsește deasupra diagonalei principale.

15. În matricea pătrată a , cu 10 linii și 10 coloane, elementul $a[3][2]$ se găsește sub diagonala secundară.

16. Secvența următoare de program calculează minimul elementelor de pe diagonala principală a unei matrice pătrate a , de dimensiune $n \times n$:

```
int n, i, x, a[50][50];
```

```
for (i=1; i<n; i++)
```

```
if (a[i][i]<a[0][0])
```

```
{x=a[i][i]; a[i][i]=a[0][0]; a[0][0]=x;}
```

17. Algoritmul de căutare binară se folosește pentru a sorta elementele unui vector.

Alegeți:

1. Puteți să aflați informații dintr-o structură de date, prin operația de:

- a) actualizare c) consultare
b) sortare d) divizare

2. Dintr-o structură de date obțineți mai multe structuri de date, prin operația de:

- a) divizare c) redenumire
b) concatenare d) actualizare

3. Se consideră următoarea matrice, stocată în memoria internă a calculatorului în ordinea coloanelor. Elementul cu numărul de ordine 5 este:

a	b	c	d	e
f	g	h	i	j
k	l	m	n	o

- a) e
b) g
c) h

4. Se consideră următoarea matrice cu numele *alfa*, stocată în memoria internă a calculatorului în ordinea liniilor. Elementul *alfa*(2,3) are valoarea:

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15

- a) 12
b) 8
c) 6

5. Care dintre următoarele variante nu reprezintă declararea corectă a două variabile de memorie de tip întreg?

- a) `int x,y;` c) `typedef intreg int; intreg x,y;`

b) unsigned x; long y; d) int x[2];

6. Care dintre următoarele secvențe de instrucțiuni determină, în variabila reală *max*, cel mai mare element dintr-un vector *a* cu numere întregi?

- a) `max = 0;`
 for (`i=0; i<n; i++`)
 if (`max<a[i]`) `max=a[i];`
- b) `max = a[0];`
 for (`i=1; i<=n; i++`)
 if (`max<a[i]`) `max=a[i];`
- c) `max = a[0];`
 for (`i=1; i<n; i++`)
 if (`max>a[i]`) `max=a[i];`
- d) `max = a[n-1];`
 for (`i=1; i<n; i++`)
 if (`max<a[i-1]`) `max=a[i-1];`

7. Care dintre următoarele secvențe de instrucțiuni determină, în variabila întreagă *s*, suma elementelor de sub diagonală secundară a unei matrice pătrate *a*, cu numere întregi, cu dimensiunea $n \times n$?

- a) `s=0;`
 for (`i=0; i<n; i++`)
 for (`j=i+1; j<n; j++`)
 `s+=a[i][j];`
- b) `s=0;`
 for (`i=0; i<n; i++`)
 for (`j=i; j<n; j++`)
 `s+= a[i][j];`
- c) `s=0;`
 for (`i=0; i<n; i++`)
 for (`j=0; j<=i-1; j++`)
 `s+= a[i][ n-j-1];`
- d) `s=0;`
 for (`i=1; i<n; i++`)
 for (`j=1; j<=i; j++`)
 `s+= a[i][n-j];`

8. Secvența următoare de program realizează:

```
int n,i,x,a[20];
.....
for (i=0;i<n-1;i++)
    if (a[i+1] < a[i]) {x=a[i]; a[i]=a[i+1]; a[i+1]=x;}
```

- a) sortarea crescătoare a vectorului folosind metoda selectării directe;
 b) calcularea valorii minime a elementelor în primul element al vectorului;
 c) calcularea valorii maxime a elementelor în ultimul element al vectorului;
 d) sortarea descrescătoare a vectorului folosind metoda inserării directe.

9. Secvența următoare de program realizează:

```
int n,i,x,p,q,a[20][20];
.....
for (i=0;i<n;i++)
    (x=a[i][p]; a[i][p]=a[i][q]; a[i][q]=x)
```

- a) sortarea elementelor de pe linia *i* folosind metoda selectării directe;
 b) interschimbarea coloanei *p* cu coloana *q*;
 c) interschimbarea liniei *p* cu linia *q*;
 d) sortarea elementelor de pe linia *i* folosind metoda bulelor.

10. Dacă vectorii *a* și *b* se folosesc pentru a memora elementele a două mulțimi, *A* și *B*, secvența următoare de program realizează, în vectorul *c*:

```
int i,j,k,n,m,a[50],b[50],c[50];
.....
for (k=0,i=0;i<n;i++)
    for (j=0;j<m;j++)
        if (a[i]==b[j]) {c[k]=a[i]; k++;}
```

- a) $A \cup B$ b) $A \cap B$ c) $A - B$ d) $B - A$

Rezolvați:

1. Un vector conține maxim 50 de numere întregi. Lungimea vectorului și elementele sale se citesc de la tastatură. Să se afișeze câte dintre elemente au valoarea mai mare decât media aritmetică a elementelor vectorului.
2. Un vector conține maxim 50 de numere întregi. Lungimea vectorului și elementele sale se citesc de la tastatură. Să se afișeze câte dintre elemente au valoarea egală cu suma elementelor vecine.
3. Un vector conține maxim 50 de numere întregi. Lungimea vectorului și elementele sale se citesc de la tastatură. Să se afișeze suma elementelor impare aflate pe pozițiile pare.
4. Doi vectori conțin fiecare maxim 50 de numere reale. Lungimile vectorilor și elementele lor se citesc de la tastatură. Să se afișeze câte dintre elementele primului vector sunt strict mai mari decât toate elementele celui de al doilea vector.
5. Se citesc de la tastatură un număr natural n și două valori reale x și y . Să se genereze recursiv într-un vector primii n termeni ai șirului: 1, 1, 2, 3, 3, 4, 5, 5, Să se afișeze câți termeni ai șirului sunt mai mari decât x și mai mici decât y și care sunt acești termeni.
6. Să se verifice că un vector cu n componente numere întregi are proprietatea: elementele sale sunt cifre binare alternative ($\{0, 1, 0, 1, 0, 1, \dots\}$ sau $\{1, 0, 1, 0, 1, 0, \dots\}$).
7. Se consideră trei vectori cu elemente numere întregi: a , b și c , cu lungimea n . Lungimea vectorilor și elementele vectorilor a și b se citesc de la tastatură. Să se afișeze vectorul c ale cărui elemente se generează astfel:
 - a) $c[i]$ este modulul diferenței $a[i] - b[i]$;
 - b) $c[i]$ este maximul dintre $a[i]$ și $b[i]$;
 - c) $c[i]$ este minimul dintre $a[i]$ și $b[n-i-1]$.
8. Să se genereze în doi vectori a și b , și apoi să se afișeze, termenii șirurilor a_n și b_n , definite recursiv astfel:

$$\begin{array}{lcl} a_n = (a_{n-1} + b_{n-1})/2 & & b_n = (a_{n-1} + 2 \times b_{n-1})/3 \\ \hline a_1 = (a_0 + b_0)/2 & & b_1 = (a_0 + 2 \times b_0)/3 \end{array}$$
 unde a_0, b_0 și n se citesc de la tastatură.
9. Un vector a conține maxim 100 de numere întregi. Lungimea vectorului și elementele sale se citesc de la tastatură. Să se copieze într-un vector b toate elementele din vectorul a care au valoarea egală cu codul ASCII al unei litere. Să se sorteze crescător vectorul b folosind metoda bulelor și să se afișeze literele din vectorul b .
10. Se consideră un vector a cu elemente numere întregi, cu lungimea n , n fiind un număr par. Lungimea vectorului, un număr natural k ($1 \leq k \leq n-2$) și elementele sale $a[0], a[1], a[2], \dots, a[n-1]$ se citesc de la tastatură. Să se rearanjeze elementele vectorului și să se afișeze astfel:
 - a) $a[n-1], a[0], a[1], a[2], \dots, a[n-2]$;
 - b) $a[1], a[2], a[3], \dots, a[n-1], a[0]$;
 - c) $a[1], a[0], a[3], a[2], \dots, a[n-1], a[n-2]$;
 - d) $a[n-2], a[n-1], a[n-4], a[n-3], \dots, a[2], a[3], a[0], a[1]$;
 - e) $a[k], a[k+1], a[k+2], \dots, a[n-2], a[n-1], a[0], a[1], \dots, a[k-2], a[k-1]$.

11. Se
Lun
vect
afiș
fost
num
12. Se c
a) A
13. Se c
14. Se c
că n
15. Să s
citit c
16. Să s
poate
17. Să s
care
18. Se c
rear
nume
19. Să s
tură)
20. Să s
a) si
b) si
c) si
d) si
21. Să s
sens in
22. Se con
 $n \times n$ ele
copiez
a) lini
b) col
c) în s
23. Se con
lorile pe
de la ta
matrice
24. Se cons
lorile pe
de la ta
liniile p și

11. Se consideră doi vectori cu elemente numere întregi a și b , cu lungimea n . Lungimea vectorilor și elementele vectorului a se citesc de la tastatură. În vectorul b în elementul $b[i]$ se calculează suma cifrelor elementului $a[i]$. Să se afișeze numărul care are cea mai mare sumă a cifrelor și al cătelea număr citit a fost. Dacă sunt mai multe numere care au suma cifrelor maximă, să se afișeze numărul de ordine la citire pentru fiecare dintre ele.
12. Se consideră două mulțimi A și B . Să se verifice dacă:
a) $A \subset B$; b) $A \supset B$; c) $A = B$.
13. Se consideră două mulțimi A și B . Să se calculeze produsul cartezian $A \times B$.
14. Se citește de la tastatură un număr întreg cu maxim 20 de cifre. Să se verifice dacă numărul este palindrom (**Indicație**. Cifrele numărului vor fi citite într-un vector.)
15. Să se memoreze într-un vector cifrele reprezentării în baza 2 ale unui număr n citit de la tastatură și să se afișeze numărul binar obținut.
16. Să se memoreze într-un vector n cifre. Să se afișeze cel mai mic număr care se poate obține cu aceste cifre.
17. Să se genereze toate numerele cu n cifre (n se citește de la tastatură și $n \leq 20$) care sunt egale cu suma cuburilor cifrelor lor.
18. Se consideră un vector a cu elemente numere întregi, cu lungimea n . Să se rearanjeze elementele vectorului astfel încât numerele pare să fie scrise înaintea numerelor impare.
19. Să se verifice că o matrice pătrată cu dimensiunea $n \times n$ (n se citește de la tastatură) este matricea zero (matricea care are toate elementele egale cu 0).
20. Să se verifice dacă o matrice pătrată cu dimensiunea $n \times n$ este:
a) simetrică față de axa orizontală,
b) simetrică față de axa verticală,
c) simetrică față de diagonala principală,
d) simetrică față de diagonala secundară.
21. Să se afișeze elementele de pe conturul matricei, parcurgerea lor făcându-se în sens invers trigonometric.
22. Se consideră o matrice pătrată a cu dimensiunea $n \times n$ (n impar) și un vector b cu $n \times n$ elemente. Numărul n și elementele matricei se citesc de la tastatură. Să se copieze în vectorul b elementele matricei a parcurse:
a) linie cu linie,
b) coloană cu coloană,
c) în spirală (pornind de la elementul $a[0][0]$ în sens invers trigonometric).
23. Se consideră o matrice a cu n linii și m coloane cu elemente numere reale. Valorile pentru n și m și elementele matricei se citesc de la tastatură. Se mai citesc de la tastatură două numere întregi p ($1 \leq p \leq n$) și q ($1 \leq q \leq m$). Să se elimine din matrice linia p și coloana q . Să se afișeze matricea obținută.
24. Se consideră o matrice a cu n linii și m coloane cu elemente numere reale. Valorile pentru n și m și elementele matricei se citesc de la tastatură. Se mai citesc de la tastatură două numere întregi p ($1 \leq p \leq n$) și q ($1 \leq q \leq n$). Să se interschimbe liniile p și q ale matricei. Să se afișeze matricea obținută.

25. Se consideră o matrice a cu n linii și m coloane cu elemente numere reale. Valorile pentru n și m și elementele matricei se citesc de la tastatură. Se mai citesc de la tastatură două numere întregi p ($1 \leq p \leq m$) și q ($1 \leq q \leq n$). Să se interschimbe coloanele p și q ale matricei. Să se afișeze matricea obținută.
26. Se consideră o matrice a cu n linii și m coloane cu elemente numere reale. Valorile pentru n și m și elementele matricei se citesc de la tastatură. Să se bordonze matricea cu coloana $m+1$, ale cărei elemente $a[i][m+1]$ au ca valoare media aritmetică a celor m elemente din linia i , și cu linia $n+1$, ale cărei elemente $a[n+1][j]$ au ca valoare media aritmetică a celor n elemente din coloana j . Să se afișeze matricea obținută.
27. Se consideră o matrice a cu n linii și m coloane cu elemente numere reale. Valorile pentru n și m și elementele matricei se citesc de la tastatură. Să se afișeze numărul liniei și numărul coloanei pe care suma elementelor este maximă.
28. Se consideră o matrice pătrată cu dimensiunea $n \times n$ și un vector cu n elemente. Numărul n și elementele matricei și ale vectorului se citesc de la tastatură. Să se verifice dacă elementele vectorului formează o linie sau o coloană a matricei. În caz afirmativ să se afișeze un mesaj în care să se precizeze numărul liniei și/sau al coloanei.
29. Găsiți metoda adecvată prin care să memorați coordonatele carteziene în spațiu (x_i, y_i, z_i) a n vectori, astfel încât să puteți implementa un algoritm cât mai eficient care să afișeze vectorii perpendiculari (vectorii al căror produs scalar este 0). Afișați vectorii perpendiculari.
30. Se consideră o matrice pătrată cu elemente numere întregi cu dimensiunea $n \times n$. Numărul n și elementele matricei se citesc de la tastatură. Să se afișeze:
 - a) elementele situate deasupra diagonalei principale;
 - b) elementele situate sub diagonala secundară.
31. Se consideră o matrice pătrată cu elemente numere întregi cu dimensiunea $n \times n$. Numărul n și elementele matricei se citesc de la tastatură. Să se afișeze:
 - a) suma elementelor situate deasupra diagonalei principale;
 - b) suma elementelor situate sub diagonala secundară;
 - c) simetrica matricei față de axa verticală care trece prin centrul matricei,
 - d) simetrica matricei față de axa orizontală care trece prin centrul matricei,
 - e) simetrica matricei față de diagonala principală,
 - f) simetrica matricei față de diagonala secundară.
32. Se consideră o matrice pătrată cu elemente numere întregi cu dimensiunea $n \times n$. Numărul n și elementele matricei se citesc de la tastatură. Să se ordoneze:
 - a) crescător, elementele de pe diagonala principală, folosind metoda selecției directe;
 - b) descrescător, elementele de pe diagonala secundară, folosind metoda bulelor;
 - c) crescător, elementele de pe linia p , folosind metoda inserării directe (p se citește de la tastatură);
 - d) descrescător, elementele de pe coloana q , folosind metoda inserării rapide (q se citește de la tastatură).

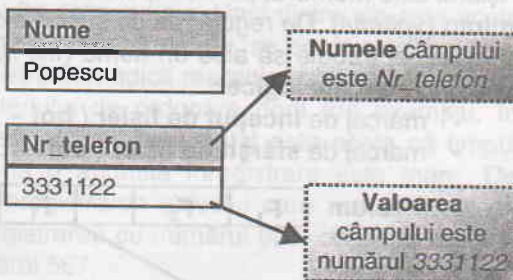
5.5. Fișierele

Să presupunem că trebuie să folosiți o structură de date pentru a păstra informațiile despre elevii unei clase: numele elevilor, mediile semestriale la disciplinele de studiu și mediile anuale. Datele prin care se înregistrează numele și prenumele elevilor sunt de tip șir de caractere, iar cele care înregistrează mediile sunt de tip real. Tablourile de memorie sunt însă structuri de date omogene. Pentru a înregistra aceste date în vederea prelucrării, ar trebui să se folosească două tablouri de memorie: unul pentru numele și prenumele elevilor (*a*), iar altul pentru medii (*b*). Ambele tablouri au același număr de linii, egal cu numărul elevilor din clasă. Legătura între cele două tablouri se face prin indice: elevului cu numele *a[i][1]* și prenumele *a[i][2]* îi corespund mediile *b[i][j]*. Procesul de prelucrare se complică din cauză că trebuie folosite două tablouri și trebuie avută grijă să se respecte corespondența între indici. În plus, o astfel de structură de date este temporară. De fiecare dată când se folosește programul pentru calcularea mediilor, trebuie introduse din nou toate datele de intrare. În astfel de aplicații se preferă un alt tip de structură de date, și anume **fișierul de date**.

La fel ca și o bibliotecă, memoria externă poate fi folosită pentru a **păstra informația**. Metoda folosită este de a grupa informațiile în colecții de date numite fișiere.

Pentru a înțelege modul în care lucrează **fișierele de date**, trebuie înțeles mai întâi modul în care sunt organizate datele în fișiere. Pentru a fi prelucrate de calculator sau păstrate în memoria externă, datele sunt organizate de obicei în grupuri. Fiecare grup este mai complex decât precedentul său:

- ✓ **Câmpul** (*field*) conține un set de caractere care au o legătură între ele și care formează împreună o entitate de informație. El păstrează aceeași categorie de informație pentru tot fișierul. De exemplu, în școală, într-o evidență a elevilor, numele este un câmp, prenumele alt câmp, adresa alt câmp, numărul de telefon alt câmp etc. Fiecare câmp este caracterizat, ca și data elementară, prin nume, tip și valoare. În plus, în unele cazuri trebuie precizată și lungimea câmpului.



- ✓ **Înregistrarea** (*record*) este o colecție de câmpuri care au o legătură între ele, deoarece în multe dintre aplicații sunt necesare mai multe câmpuri care să fie prelucrate împreună. De exemplu, toate informațiile despre un elev, inclusiv un număr de identificare a elevului, formează o înregistrare. Pentru a permite operații de prelucrare mai complexe, numele elevului nu este înregistrat într-un singur câmp ci în trei câmpuri: nume, inițiala tatălui și prenume. Principala caracteristică a înregistrării este aceea că ea conține informații care vor fi prelucrate împreună (în exemplu, informații care se referă la un elev). Înregistrarea poate conține un câmp sau mai multe câmpuri și poate avea diferite lungimi. Dacă ea este formată dintr-un număr fix de câmpuri de lungime fixă, va avea o lungime fixă, altfel lungimea sa este variabilă.

- ✓ **Fișierul (file)** este o colecție de înregistrări legate între ele. Toate înregistrările despre elevii școlii pot forma un fișier. Prelucrarea datelor din fișier se poate face secvențial sau în acces direct. Pentru a avea acces direct la o înregistrare, se poate alege unul dintre câmpuri pentru a identifica unic înregistrarea (de exemplu, numărul matricol al elevului).

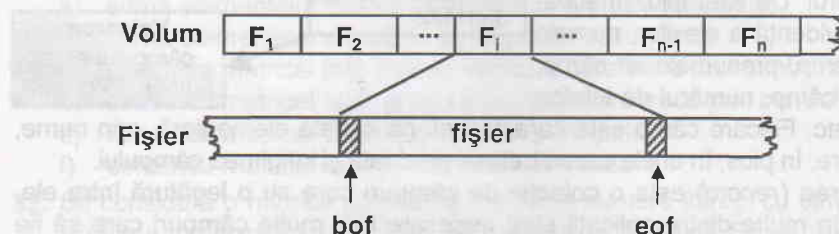
Înregistrarea poate fi și ea considerată o structură de date, în care elementele structurii sunt câmpurile. Accesul la elementele structurii se face prin numele câmpului. Din această cauză sistemul trebuie să aibă informații explicite despre câmpuri, adică despre elementele care compun structura. Aceste informații se referă la numele câmpului, tipul datelor memorate în câmp și lungimea câmpului.

Așadar, fișierul este o structură de date în care elementele structurii sunt înregistrările. Identificarea unui element se face printr-un număr de ordine numit **numărul înregistrării (record number)**. De exemplu, dacă în școală sunt 1500 de elevi, fișierul de date *elevi* va avea 1500 de înregistrări, câte o înregistrare pentru fiecare elev, cărora li se va atribui un număr unic de la 1 la 1500, în ordinea în care au fost scrise în fișier. Înregistrarea cu numărul 9 înseamnă a noua înregistrare din fișier. **Înregistrarea curentă** este înregistrarea care se prelucurează la un moment dat.

Fișierul (file) este o structură de date externă formată dintr-o mulțime ordonată de înregistrări, ordonarea făcându-se după numărul înregistrării (un număr atribuit unic înregistrării, atunci când aceasta este adăugată la fișier).

Fișierul este memorat pe un suport de informație. Suportul de informație se numește **volum (volume)**. De regulă, pe un suport există mai multe fișiere. Pentru a identifica un fișier, el trebuie să aibă un nume (*file name*). Delimitarea fișierului pe suport se face prin **marcaje logice** :

- ✓ marcaj de **început de fișier** (*bof* – *Begin Of File*),
- ✓ marcaj de **sfârșit de fișier** (*eof* – *End Of File*)



Limbajele care permit prelucrarea fișierelor pun la dispoziția utilizatorului două funcții prin care se poate testa dacă înregistrarea curentă este marcajul de început sau de sfârșit de fișier.

Prelucrarea datelor dintr-un fișier presupune următoarele operații:

1. Se **deschide (open)** fișierul de date. În urma acestei operații i se atribuie o zonă de lucru în memoria internă, în care se transferă de pe disc înregistrările care se prelucurează. Tot în această zonă se păstrează informații despre starea fișierului

de date: numărul total de înregistrări, numărul înregistrării curente (înregistrarea care se prelucreează la un moment dat) etc.

2. Se **exploatează** fișierul de date executându-se diferite operații de actualizare (adăugare, ștergere, modificare) sau de consultare. Aceste operații se realizează la nivel de înregistrare, adică, la un moment dat, nu se poate șterge, adăuga sau modifica decât câte o singură înregistrare. Selectarea înregistrării asupra căreia acționează operația de actualizare sau consultare se face prin mecanismul **indicatorului de înregistrare** care este o variabilă de memorie în care se păstrează **numărul înregistrării curente**. În general, limbajele folosite pentru dezvoltarea programelor de exploatare a fișierelor de date pun la dispoziție funcții prin care furnizează numărul înregistrării curente și numărul total de înregistrări din fișier. Exploatarea fișierului se desfășoară astfel:
 - ✓ Se caută în fișierul de pe disc înregistrarea care urmează să fie prelucrată (actualizată sau consultată). Identificarea ei se poate face după numărul ei.
 - ✓ Dacă se găsește înregistrarea, ea este copiată într-o zonă din memoria internă alocată de sistem pentru prelucrarea fișierului.
 - ✓ Se prelucreează înregistrarea.
 - ✓ Dacă operația de prelucrare a fost o operație de actualizare, înregistrarea este copiată din memoria internă pe disc, înlocuind vechea înregistrare.
3. Se **închide** (*close*) fișierul de date când se eliberează zona de memorie alocată pentru lucru.

Prelucrarea înregistrărilor unui fișier se poate face în acces secvențial sau în acces direct:

- ✓ **Accesul secvențial** prezintă mai multe dezavantaje. Unul dintre ele este acela că, pentru reorganizarea fișierului într-o anumită ordine, se consumă timp pentru operația de sortare fizică a înregistrărilor, adică rearanjarea înregistrărilor pe suportul de informație conform criteriului de ordonare dorit (de exemplu, în ordinea alfabetică a numelui elevilor). Un alt dezavantaj este acela că timpul necesar pentru a asigura accesul la o anumită înregistrare este mare. De exemplu, pentru a avea acces la înregistrarea elevului care are numărul de ordine 567, căutarea începe cu înregistrarea cu numărul 000, continuă cu 001, 002 ș.a.m.d. până se ajunge la numărul 567.
- ✓ **Accesul direct** prezintă avantajul că poate localiza rapid o înregistrare, în funcție de valoarea unui câmp stabilit pentru identificare. Accesul direct la înregistrările fișierului presupune însă un anumit mod de organizare a datelor pe suport, care consumă mai mult suport de memorare decât cel secvențial. Se recomandă folosirea accesului secvențial, atunci când aceeași operație se execută asupra unui număr mare de înregistrări din fișier și în cazul listării unor rapoarte, iar accesul direct, în cazul în care se prelucreează un număr mic de înregistrări care vor fi localizate după valoarea unui anumit câmp.

Așadar, **fișierul este o zonă de memorie externă (mai multe sectoare de disc ce nu sunt obligatoriu aranjate continuu) căreia i se atribuie un nume și care permite memorarea mai multor înregistrări. Această colecție de înregistrări poate fi tratată ca un tot unitar sau ca ansamblu de elemente independente.**

Fișierul este o structură de date omogenă (este format din elemente de același tip, adică din înregistrări), cu acces direct sau secvențial (în funcție de modul în care a fost organizat pe suportul de informație), externă, permanentă și dinamică.

Așadar, implementarea unui fișier se face:

- ✓ **Din punct de vedere logic.** Fișierul este o structură de date omogenă cu elemente de același tip numite înregistrări. Este o structură liniară, în care fiecare înregistrare, cu excepția ultimei, are un succesor unic, localizarea unui element în cadrul structurii făcându-se după numărul de ordine al înregistrării. Scrierea înregistrărilor în fișier se face în ordinea în care sunt scrise de la tastatură.
- ✓ **Din punct de vedere fizic.** Fișierului i se alocă memorie pe suportul de informație, de către sistemul de operare. Alocarea se face dinamic, în funcție de necesitățile fișierului, în zonele libere ale suportului, sub forma unor porțiuni de suport numite unități de alocare. Evidența unităților de alocare care aparțin aceluiași fișier este în sarcina sistemului de operare.

Asupra fișierelor se pot executa toate operațiile prezentate la pagina 157. Atunci când fișierul este copiat, mutat, redenumit sau șters, ansamblul de înregistrări este tratat ca un tot unitar, iar când se execută operații de actualizare sau consultare, înregistrarea este tratată ca o structură de date independentă.

În limbajul C++ sunt implementate două tipuri de fișiere:

- ✓ **Fișiere text.** Conțin numai caractere reprezentate în codul ASCII, iar înregistrarea este o linie de text.
- ✓ **Fișiere binare.** Înregistrările sunt de același tip (fie date elementare de tip `int`, `float` etc.), fie structuri de date organizate sub forma unei colecții de câmpuri.

Cunoștințele dobândite până în acest moment nu vă permit lucrul cu fișierele binare care conțin înregistrări structurate în câmpuri.

5.6. Implementarea fișierelor text în limbajul C++

Fișierul text are următoarele caracteristici:

- ✓ Este o **colecție de linii de text**. Liniile de text au lungimi diferite. Fiecare linie de text reprezintă o înregistrare. Așadar, fișierul text este un fișier cu înregistrări de lungime variabilă.
- ✓ În cadrul unei înregistrări, entitatea prelucrată este octetul. Conținutul unui octet este interpretat ca fiind codul ASCII al unui caracter.
- ✓ Liniile de text (înregistrările de lungime variabilă) sunt separate de marcajul **newline** (`'\n'`) care se generează la apăsarea tastei **Enter**.
- ✓ Fișierul de text se termină prin marcajul **eof**¹ (**end of file**) care se generează prin apăsarea tastelor **Ctrl+Z** și care are codul ASCII $1A_{(16)}=26_{(10)}$.
- ✓ **Entitatea prelucrată într-o operație de citire sau de scriere este octetul.**
- ✓ Selectarea octetului asupra căruia acționează operația de citire sau de scriere se face prin mecanismul **pointerului către octet**, care este o variabilă de me-

¹ Marcajele **newline** și **eof** sunt inserate automat în fișier la crearea lui prin operația de scriere.

5.6.1. FI

morie în care se păstrează **adresa octetului curent**. Putem să ne închipuim acest pointer ca pe un cursor care se deplasează cu o poziție (cu un octet) după fiecare operație de citire sau scriere.

- ✓ Fișierele text pot fi citite și modificate cu orice editor de texte.

Numele fișierului. Fișierul are două nume:

- ✓ **Numele extern** sau **numele fizic**. Este numele sub care este cunoscut de sistemul de operare, adică numele cu care este înregistrat în director.
- ✓ **Numele intern** sau **numele logic**. Este numele sub care este cunoscut în cadrul programului, adică identificatorul care i s-a atribuit în program.

Starea fișierului poate fi:

1. **Închis** (*close*). Fișierul nu poate fi folosit (nu este disponibil pentru operațiile de citire și de scriere). Este starea la începutul execuției programului.
2. **Deschis** (*open*). Fișierul poate fi folosit pentru citire sau scriere. El poate fi:
 - ✓ deschis pentru citire,
 - ✓ deschis pentru scriere.

Poziția prelucrată din fișier la:

1. **Citare:**
 - ✓ la deschidere, pointerul indică poziția primului octet din fișier;
 - ✓ fiecare operație de citire deplasează pointerul la octetul următor; acest proces de avansare poate continua până când pointerul va fi poziționat pe marcajul de sfârșit de fișier (*eof*).
2. **Scriere:**
 - ✓ la deschidere, pointerul indică primul caracter din fișier, la operația de creare, sau marcajul *eof*, la operația de adăugare;
 - ✓ fiecare operație de scriere adaugă un nou octet la fișier, crescând dimensiunea acestuia, și deplasează pointerul cu o poziție.

Pentru operațiile de citire și de scriere se construiesc, cu ajutorul limbajului, fluxuri de date. Fluxurile de date sunt conectate la fișiere. Există două **fișiere text standard**, care au numele logic:

- ✓ **cin** – este un fișier text de intrare (asupra lui se poate executa numai operația de citire) și este atribuit tastaturii.
- ✓ **cout** – este un fișier text de ieșire (asupra lui se poate executa numai operația de scriere) și este atribuit monitorului.

5.6.1. Fluxuri de date pentru fișiere text

Dacă declararea fluxurilor de date standard este implementată în bibliotecile sistemului (fișierul antet **iostream.h**), pentru celelalte fișiere trebuie declarat fluxul de date. În programele în care lucrați cu fluxuri de date pentru fișiere trebuie să includeți fișierul antet **fstream.h** cu instrucțiunea pentru preprocesor:

```
#include <fstream.h>
```

Declararea unui flux de date pentru un fișier se face cu instrucțiunea:

```
fstream nume_logic (nume_fizic, mod_de_deschidere);
```


unde **nume_logic** este un identificator care se va folosi în program pentru a face referiri la acel fișier, **nume_fizic** este o constantă de tip șir de caractere care reprezintă numele fizic al fișierului (identificatorul său complet pentru sistemul de operare, care va conține inclusiv calea de director, dacă fișierul nu se găsește în directorul curent), iar **mod_de_deschidere** este o construcție de forma:

ios::constantă_întreagă

sau mai multe construcții de această formă legate de operatorul **sau logic pe biți** (**|**), unde **ios::** permite accesul la aceste constante.

Această instrucțiune realizează următoarele:

- ✓ creează fluxul de date între fișier și program,
- ✓ deschide fișierul, și
- ✓ conectează fișierul la fluxul de date.

Prin **constantele de sistem** folosite pentru **mod_de_deschidere** stabiliți operațiile care se pot executa cu fișierul conectat la acest flux. Pentru aceste constante există implementate, în biblioteca de fluxuri de date, și constante simbolice. Ele au următoarea semnificație:

Constanta	Constanta simbolică	Operații permise
0x01	in	Fișierul se deschide pentru citire.
0x02	out	Fișierul se deschide pentru scriere; dacă fișierul există, scrierea se face la începutul lui (vechiul conținut se pierde), iar dacă fișierul nu există, se creează.
0x04	ate	După deschidere (pentru scriere sau pentru citire) se face un salt la sfârșitul fișierului.
0x08	app	Fișierul se deschide pentru adăugare (scriere la sfârșitul fișierului); dacă fișierul nu există, se creează.
0x10	trunc	Dacă fișierul există, se deschide pentru scriere la începutul lui (datele din fișier se pierd).
0x20	nocreate	Fișierul este deschis numai dacă există (dacă nu există, nu va fi creat).
0x40	noreplace	Dacă fișierul nu există, este deschis pentru creare, altfel nu este deschis.
0x80	binary	Se deschide un fișier binar.

De exemplu, cu instrucțiunile:

```
fstream f1("c:\\alfa.txt", ios::in);
fstream f2("c:\\beta\\alfa.txt", ios::app);
```

se deschid două fluxuri de date pentru două fișiere care vor fi conectate la aceste fluxuri: un flux pentru fișierul *alfa.txt* care se găsește în directorul rădăcină al discului *C*, referirea lui în program făcându-se cu identificatorul *f1*, și un flux pentru fișierul *alfa.txt* care se găsește în subdirectorul *gama* din directorul rădăcină al discului *C*, referirea lui în program făcându-se cu identificatorul *f2*. Fișierul *f1* este deschis numai pentru operația de citire, iar fișierul *f2* numai pentru operația de adăugare.



Deoarece caracterul \ este folosit pentru secvențele **escape**, pentru a-l putea include într-un șir de caractere (cum este cazul identificatorului unui fișier în care caracterul \ este folosit ca separator în calea de directoare), el trebuie scris de două ori.

Cu instrucțiunea:

```
fstream f1("alfa.txt", ios::in), f2("beta.txt", ios::out);
```

se deschid două fluxuri de date pentru două fișiere care vor fi conectate la aceste fluxuri: un flux pentru fișierul *alfa.txt*, referirea lui în program făcându-se cu identificatorul *f1*, și un flux pentru fișierul *beta.txt*, referirea lui în program făcându-se cu identificatorul *f2*. Ambele fișiere se găsesc în directorul curent. Fișierul *f1* este deschis numai pentru operația de citire, iar fișierul *f2* numai pentru operația de scriere.

În prelucrarea unui fișier text vă sunt utile următoarele **funcții de sistem**:

Funcția	Realizează
eof()	Se folosește pentru a detecta sfârșitul de fișier. Dacă valoarea citită în poziția curentă este codul ASCII pentru sfârșit de fișier, funcția furnizează valoarea 1, altfel furnizează valoarea 0.
close()	Închide un fișier deschis.
tellp()/tellg()	Furnizează poziția pointerului față de începutul fișierului la scriere, respectiv la citire. Rezultatul furnizat este de tip long .
seekp(x,y)/seekg(x,y)	Deplasează pointerul în poziția precizată la scriere, respectiv la citire. Parametrul x reprezintă deplasarea față de o poziție de referință precizată prin parametrul y (care este o constantă de sistem și care poate avea una dintre valorile: beg – început de fișier, cur – poziția curentă sau eof – sfârșit de fișier). Parametrul x este de tip long și poate avea valori pozitive (deplasarea se face la dreapta poziției de referință) sau negative (deplasarea se face la stânga poziției de referință).
clear(x)	Stabilește starea fluxului de date la valoarea precizată. Parametrul x este de tip întreg. Dacă are valoarea 0, starea fluxului este bună (se poate executa o operație de citire). Dacă are valoarea 1 înseamnă că s-a ajuns la sfârșitul fișierului.
get(x,n,'d')	Extrage un caracter dintr-un șir de caractere x până se întâmplă unul dintre evenimentele următoare: a întâlnit marcajul de sfârșit de linie sau marcajul de sfârșit de fișier, au fost citite n caractere sau a citit caracterul folosit ca delimitator 'd' .
get()	Extrage următorul caracter sau marcajul de sfârșit de fișier.
getline(x, n, 'd')	Are același efect ca și get(x,n,'d') , cu deosebirea că extrage și delimitatorul de sfârșit de linie ('/n' sau 'd').

Observație. Apelarea acestor funcții se face cu o construcție de forma:

```
nume_logic_fișier . nume_funcție(...);
```




Funcțiile din biblioteca pentru fluxurile de date sunt implementate folosind programarea orientată pe obiecte. Din această cauză, la apelarea lor folosiți operatori a căror semnificație nu a fost definită și nu poate fi definită, neavând cunoștințe despre programarea orientată pe obiecte (cum sunt de exemplu operatorii `.` și `::`). Rețineți doar că acești operatori au prioritate maximă. Mai rețineți că accesul la orice constantă de sistem se face prin construcția:

ios::cor stantă_întreagă

În prelucrarea unui fișier text vă mai sunt utile următoarele **macrocomenzi**, care nu acționează asupra fluxului de date al fișierului, ci chiar asupra fișierului. Din această cauză, pentru a le putea folosi, fișierul trebuie să fie închis. Fișierul header al acestor macrocomenzi este **stdio.h**

Funcția	Realizează
rename(x,y)	Redenumeste un fișier. Parametrii x și y sunt de tip șir de caractere și reprezintă: x numele vechi, iar y numele nou.
remove(x)	Șterge un fișier. Parametrul x este de tip șir de caractere și reprezintă numele fișierului.

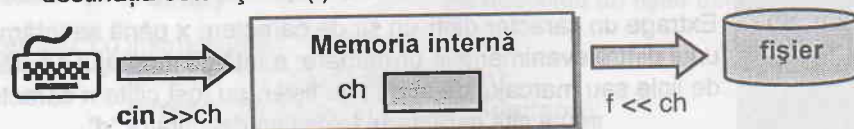
Apelarea lor se face ca în cazul celorlalte funcții și nu ca a celor din biblioteca de fluxuri.

După ce ați creat fluxul de date pentru un fișier, îl puteți exploata folosind **operatorul de intrare (>>)** și **operatorul de ieșire (<<)**.

Operația de scriere în fișier

Pentru citirea datelor de la tastatură și scrierea lor într-un fișier text se folosesc două fluxuri de date:

- Fluxul de date al tastaturii (cin >> ch)** – pentru citire de la tastatură:
 - ✓ sursa este tastatura (fișierul standard **cin**),
 - ✓ destinația este o variabilă de memorie (**ch**).
- Fluxul de date al fișierului (f << ch)** – pentru scriere în fișier:
 - ✓ sursa este variabila de memorie (**ch**),
 - ✓ destinația este fișierul (**f**).



Secvența de instrucțiuni folosită este:

```
char ch, nume[20];
cout<<"nume fișier ="; cin>>nume;
fstream f(nume, ios::out);
while (cin>>ch) f<<ch;
f.close();
```

Observații:

1. În declararea fluxului de date pentru un fișier cu **fstream** pentru numele logic al fișierului se poate folosi și o variabilă de memorie de tip șir de caractere (în exemplu, variabila *nume*). În acest mod, puteți folosi programul pentru orice fișier text (numele fizic al fișierului va fi comunicat în timpul execuției programului).
2. Expresia **cin>>ch** este folosită pentru a testa momentul în care se termină execuția repetată a instrucțiunii **while**. Prin evaluarea ei se testează de fapt starea fluxului de date al tastaturii: dacă s-a citit un caracter de la tastatură, rezultatul ei va fi un număr diferit de 0 (valoarea logică *True*), iar dacă s-a citit sfârșitul de fișier generat prin apăsarea tastelor **Ctrl+Z**, ea va furniza rezultatul 0 (valoarea logică *False*).
3. Operația de **citire se face fără format**. Aceasta înseamnă că sunt ignorate caracterele albe. De exemplu, dacă de la tastatură se citește:

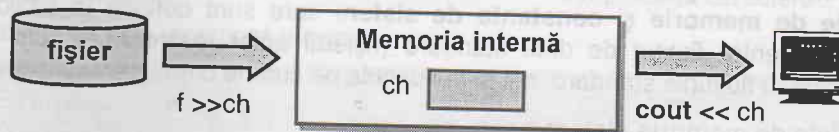
25 Spațiu 43 Enter a Enter Ctrl+Z Enter
fișierul va conține octeții cu caractere:

2	5	4	3	a	eof
---	---	---	---	---	-----

Operația de citire din fișier

Pentru citirea datelor din fișier și scrierea lor pe ecranul monitorului se folosesc două fluxuri de date:

1. **Fluxul de date al fișierului** (**f >> ch**) – pentru citire din fișier:
 - ✓ sursa este fișierul (*f*),
 - ✓ destinația este o variabilă de memorie (*ch*).
2. **Fluxul de date al monitorului** (**cout << ch**) – pentru scriere pe ecran:
 - ✓ sursa este variabila de memorie (*ch*),
 - ✓ destinația este monitorul (fișierul standard **cout**).



Secvența de instrucțiuni folosită este:

```
char ch, nume[20];
cout<<"nume fisier ="; cin>>nume;
fstream f(nume, ios::in);
while (f>>ch) cout<<ch;
f.close();
```

Observații:

1. Expresia **f>>ch** este folosită pentru a testa momentul în care se termină execuția repetată a instrucțiunii **while**. Prin evaluarea ei se testează de fapt starea fluxului de date: dacă s-a citit un caracter din fișier, rezultatul ei va fi un număr diferit de 0 (valoarea logică *True*), iar dacă s-a citit marcajul de sfârșit de fișier (**eof**), ea va furniza rezultatul 0 (valoarea logică *False*).

2. Operația de **citire se face fără format**. Aceasta înseamnă că sunt ignorate caracterele albe. De exemplu, dacă fișierul conține:

2	5	spațiu	4	3	/n	a	/n	eof
---	---	--------	---	---	----	---	----	-----

pe ecran afișarea nu se va face pe două linii: pe prima linie 25 43, iar pe a doua a, ci pe o singură linie 2543a (caracterele albe folosite pentru spațiu și pentru linie nouă sunt ignorate).

În următorul exemplu de program se execută mai întâi operația de scriere în fișier și apoi operația de citire din fișier. Pentru aceasta, se creează două fluxuri de date pentru același fișier: unul prin care fișierul va fi deschis numai pentru operația de scriere, iar altul prin care va fi deschis numai pentru operația de citire. După ce s-a scris în fișier folosind fluxul de date *f1*, fișierul trebuie închis, pentru a putea fi deschis pentru fluxul de date *f2*.

Exemplu:

```
#include <iostream.h>
#include <fstream.h>
void main()
{char ch, nume[20];
cout<<"nume fisier ="; cin>>nume;
fstream f1(nume,ios::out), f2(nume,ios::in);
while (cin>>ch) f1<<ch;
f1.close();
while (f2>>ch) cout<<ch;
f2.close();
}
```

5.6.2. Citiri și scrieri cu format

Pentru ca **citirea și scrierea** datelor să se poată face cu format, se folosesc **variabile de memorie și constante de sistem** care sunt definite în bibliotecile sistemului pentru fluxuri de date standard (fișierul antet **iostream.h**). Ele pot fi folosite atât în fluxurile standard, cât și în fluxurile pe care le creai pentru fișiere.

Variabilele de memorie sistem sunt:

Identificator	Tip	Conținut
x_precision	int	Numărul de poziții pentru partea fracționară care se afișează pentru un număr real.
x_width	int	Numărul de poziții pe care se face afișarea (lățimea câmpului pentru afișare).
x_fill	int	Caracterul de umplere folosit; acest caracter va fi scris în pozițiile libere ale câmpului pentru afișare (pozițiile în care nu este scrisă data).
x_flags	long	Conține o mască, construită la nivel de biți, pentru formatarea intrărilor și a ieșirilor. Pentru tipurile de formatare permise, sunt definite constante de sistem care pot fi atribuite variabilei.

Pentru a avea acces la aceste variabile și constante de sistem, se folosesc manipulatorii. **Manipulatorii** sunt funcții de sistem care pot fi apelate în fluxul de date, atât la citire, cât și la scriere. Pentru a avea acces la ei trebuie să includeți în fișierul sursă fișierul antet **iomanip.h**. Într-un flux de date puteți apela mai mulți manipulatori, astfel:

1. La citire:

```
cin >> manipulator_1 >> manipulator_2 >> ... >> manipulator_n >> variabilă;
sau
f >> manipulator_1 >> manipulator_2 >> ... >> manipulator_n >> variabilă;
```

2. La scriere:

```
cout << manipulator_1 << manipulator_2 << ... << manipulator_n << var|const
sau
f << manipulator_1 << manipulator_2 << ... << manipulator_n << var|const
```

unde *var|const* reprezintă un nume de variabilă de memorie sau de constantă sau o constantă.

Următorii manipulatori se folosesc numai **pentru operația de scriere**. Rolul lor este de a vă permite să atribuiți valori primelor trei variabile de sistem:

Manipulator	Tip x	Acces la	Realizează
setprecision(x)	int	x_precision	Stabilește precizia (numărul de poziții pentru partea fracționară) la afișarea unui număr real; x reprezintă numărul zecimalelor.
setw(x)	int	x_width	Stabilește numărul de poziții pe care se face afișarea; x reprezintă lățimea câmpului pentru afișare.
setfill(x)	char	x_fill	Stabilește caracterul de umplere folosit; x reprezintă caracterul.

De exemplu, prin următorul program:

```
#include <iostream.h>
#include <iomanip.h>
void main()
{float x=0.12345;
cout<<setw(5) << setprecision(2) <<setfill('0')<<x;
}
```

se stabilesc următoarele caracteristici pentru afișarea variabilei *a*: lățimea câmpului de afișare este 5, precizia de afișare este de două cifre, iar caracterul de umplere este 0. Se va afișa 00.12.

Accesul la cea de a patra variabilă de sistem **x_flags** este controlat prin manipulatorii:

Manipulator	Realizează
setiosflags (masca)	Setează biții precizați în mască (biții precizați vor avea valoarea 1) și atribuie masca variabilei x_flags .
resetiosflags (masca)	Resetează biții precizați în mască (biții precizați vor avea valoarea 0) și atribuie masca variabilei x_flags .

Masca este formată dintr-un șir de 16 biți. Prin poziționarea pe 1 a unui bit se stabilește o anumită formatare, iar prin poziționarea pe 0 a aceluiași bit se anulează formatarea. În construirea măștii se recomandă folosirea constantelor de sistem.

Pentru **constantele folosite în masca de formatare** sunt definite și constante simbolice în biblioteca de streamuri. Valoarea unei constante corespunde poziționării pe 1 a unuia dintre biți, fiecare valoare obținută corespunzând unei puteri a lui 2 în hexazecimal (1, 2, 4, 8, 10 etc.). Ele au următoarea semnificație:

Constanta	Constanta simbolică	Semnificație
0x0001	skipws	Caracterele albe care preced valoarea care urmează să fie citită sunt ignorate (sărite).
0x0002	left	Afișarea datelor se face cu aliniere la stânga.
0x0004	right	Afișarea datelor se face cu aliniere la dreapta.
0x0008	internal	Dacă lățimea câmpului pentru afișare este mai mare decât numărul de poziții ocupate de dată, afișarea semnului numărului se face cu aliniere la stânga, iar a numărului – cu aliniere la dreapta.
0x0010	dec	Se face conversia numărului în zecimal.
0x0020	oct	Se face conversia numărului în octal.
0x0040	hex	Se face conversia numărului în hexazecimal.
0x0080	showbase	Se utilizează indicatorul bazei de numerație.
0x0100	showpoint	Este forțată afișarea punctului zecimal, chiar dacă numărul real are o valoare întreagă.
0x0200	uppercase	Pentru afișarea numerelor hexazecimale se folosesc litere mari.
0x0400	showpos	Pentru valorile numerice pozitive se afișează semnul +.
0x0800	scientific	Pentru afișarea numerelor reale se folosește notația științifică.
0x1000	fixed	Pentru afișarea valorilor reale se folosește forma normală (cu semn și separatorul punct).

Pentru **formarea unei măști** cu ajutorul constantelor se pot folosi construcții de forma:

`ios::const_1 | ios::const_2 | ... | ios::const_n`

De exemplu, prin masca:

`ios::right | ios::hex | ios::uppercase`

o valoare numerică întreagă va fi convertită în hexazecimal, folosindu-se litere mari și va fi afișată prin aliniere la dreapta.

Observație: pentru conversii dintr-o bază de numerație în alta se pot folosi și manipulatori în locul măștii:

Manipulator	Realizează
<code>oct</code>	conversia în octal
<code>dec</code>	conversia în zecimal
<code>hex</code>	conversia în hexazecimal

```
#include
#include
void m
{int a
float
cin>
cout
<<x<<
cout<
```

dacă se
rânduri,
rea varia

Citirile și
folosi cî
exemplu
mul ante
datele pe

Exemplu

```
#include
#include
#include
void m
{char
cout<
fstre
while
f1.cl
while
f2.cl
```

Prin folo
prin car
Astfel, d

fișierul v

iar la citi
iar pe rân

5.6.3.

Sunt pre
puteți fol

de date

De exemplu, prin următorul program:

```
#include <iostream.h>
#include <iomanip.h>
void main()
{int a;
 float x=123;
 cin>>a;
 cout<<setw(6)<<setprecision(2)<<setiosflags(ios::showpoint)
 <<x<<endl;
 cout<<setiosflags(ios::hex)<<setiosflags(ios::uppercase)<<a;
}
```

dacă se citește de la tastatură 30 pentru variabila *a*, se va face afișarea pe două rânduri, mai întâi 123.00 (valoarea variabilei *x*) și apoi, pe rândul următor, 1E (valoarea variabilei *a* convertită la citire în hexazecimal).

Citirile și scrierile cu format pot fi folosite și în fluxurile de date ale fișierelor. Putem folosi citirea și scrierea cu format ca să nu mai fie ignorate caracterele albe (de exemplu spațiile și caracterul de linie nouă). Pentru exemplificare se modifică programul anterior care citește de la tastatură într-un fișier, și apoi afișează din acel fișier datele pe ecranul monitorului, astfel:

Exemplu:

```
#include <iostream.h>
#include <fstream.h>
#include <iomanip.h>
void main()
{char ch, nume[20];
 cout<<"nume fisier ="; cin>>nume;
 fstream f1(nume,ios::out), f2(nume,ios::in);
 while (cin>> resetiosflags(ios::skipws)>> ch) f1<<ch;
 f1.close();
 while (f2>> resetiosflags(ios::skipws)>> ch) cout<<ch;
 f2.close();
}
```

Prin folosirea manipulatorului `resetiosflags(ios::skipws)` se resetează bitul prin care sunt ignorate caracterele albe la citire și la scriere (constanta `skipws`). Astfel, dacă se citește de la tastatură:

25 43 Enter a Enter Ctrl+Z Enter

fișierul va conține octeții cu caracterele:

2	5	spațiu	4	3	/n	a	/n	eof
---	---	--------	---	---	----	---	----	-----

iar la citirea lui, pe monitor afișarea se va face pe două rânduri: pe primul rând 25 43, iar pe rândul al doilea a.

5.6.3. Aplicații cu prelucrări de fișiere

Sunt prezentate în continuare câteva aplicații care să vă exemplifice modul în care puteți folosi fluxul de date și funcțiile pentru fișiere:

Enunțul problemei 1: Să se copieze un fișier, să se șteargă vechiul fișier și să se redenumască noul fișier.

Se folosesc **identificatorii**:

- ✓ variabila *ch* pentru citirea și scrierea unui caracter și variabilele *nume1* pentru numele fișierului care se copiază, *nume2* pentru numele fișierului în care se copiază și *nume3* pentru noul nume al fișierului;
- ✓ *f1*, *f2* și *f3* pentru fluxurile de date ale fișierelor: *f1* pentru fișierul din care se copiază, *f2* pentru fișierul în care se copiază și *f3* pentru fișierul în care s-a copiat pentru a fi citit.

Un flux de date pentru un fișier se poate crea numai dacă fișierul există. Pentru a verifica dacă nu există un fișier, se verifică dacă nu există fluxul de date (condiția *!f1*).

```
#include <iostream.h>
#include <fstream.h>
#include <iomanip.h>
#include <stdio.h>
void main()
{char ch, nume1[20], nume2[20], nume3[20];
cout<<"nume fisier 1 ="; cin>>nume1;
fstream f1(nume1,ios::in);
if (!f1)
{cout<<"Fișierul " <<nume1<<" nu exista";
f1.close();}
else
{cout<<"nume fisier 2 ="; cin>>nume2;
fstream f2(nume2,ios::out);
while (f1>>resetiosflags(ios::skipws)>>ch)
f2<<resetiosflags(ios::skipws)<<ch;
f1.close(); f2.close();
remove(nume1);
cout<<"noul nume fisier ="; cin>>nume3;
rename(nume2,nume3);
fstream f3(nume3, ios::in);
while(f3>>resetiosflags(ios::skipws)>>ch) cout<<ch;
f3.close();}
}
```

Enunțul problemei 2: Să se concateneze două fișiere (al doilea fișier se va adăuga la primul).

Se folosesc **identificatorii**:

- ✓ variabila *ch* pentru citirea și scrierea unui caracter și variabilele *nume1* pentru numele fișierului în care se adaugă al doilea fișier și *nume2* pentru numele fișierului care se adaugă;
- ✓ *f1*, *f2* și *f3* pentru fluxurile de date ale fișierelor: *f1* pentru fișierul în care se adaugă, *f2* pentru fișierul care se adaugă și *f3* pentru fișierul rezultat pentru a fi citit.

```
#include <iostream.h>
#include <fstream.h>
#include <iomanip.h>
void main()
```

```

{char ch, nume1[20], nume2[20];
cout<<"nume fisier 1 ="; cin>>nume1;
cout<<"nume fisier 2 ="; cin>>nume2;
fstream f1(nume1,ios::app);
fstream f2(nume2,ios::in);
while (f2>resetiosflags(ios::skipws)>>ch)
    f1<<resetiosflags(ios::skipws)<<ch;
f1.close(); f2.close();
fstream f3(nume1,ios::in);
while (f3>>resetiosflags(ios::skipws)>>ch)    cout<<ch;
f3.close();
}

```

Enunțul problemei 3: Să se scrie într-un fișier de la tastatură linie cu linie și să se citească tot linie cu linie.

Se folosesc identificatorii:

- ✓ variabila *linie* de tip vector de caractere pentru citirea unei linii de text și variabila *nume* pentru numele fișierului;
- ✓ *f1* și *f2* pentru fluxurile de date ale fișierului: *f1* pentru scrierea în fișier și *f2* pentru citirea din fișier.

Citirile și scrierile în – și din – fișier s-au făcut în exemplele anterioare caracter cu caracter. În acest exemplu, citirea și scrierea se face linie cu linie folosindu-se funcțiile **get()** – prima variantă și **getline()** – a doua variantă. Citirea de la tastatură se face atât timp cât nu este citit caracterul **sfârșit de fișier (Ctrl+Z)**.

```

#include <iostream.h>
#include <fstream.h>
#include <iomanip.h>
void main()
{char linie[100], nume[20];
cout<<"nume fisier ="; cin>>nume;
fstream f1(nume,ios::out);
while (!cin.eof())
    {while (cin.get(linie,100))
        {f1<<linie<<endl;
        cin.get();}
    f1<<endl;}
f1.close();
fstream f2(nume, ios::in);
.....
// Varianta 1
while (f2.get(linie,100))
    {cout<<linie<<endl;
    f2.get();}
f2.close();

// Varianta 2
while (f2.getline(linie,100))
    cout<<linie<<endl;
f2.close();
}

```


Enunțul problemei 4: Să se interclaseze două fișiere. Cele două fișiere care se interclasează conțin pe câte o linie o valoare numerică întreagă, ordonate crescător.

Se folosesc **identificatorii**:

- ✓ variabilele *nr1* și *nr2* pentru citirea a câte unui număr din primul fișier, respectiv al doilea fișier, și variabilele *nume1* și *nume2*, pentru numele fișierelor sursă, și *nume3* pentru numele fișierului în care se interclasează datele;
- ✓ *f1*, *f2* și *f3* pentru fluxurile de date ale fișierelor: *f1* și *f2* pentru fișierele sursă din care se citește și *f3* pentru fișierul destinație (fișierul interclasat) în care se scrie.

Spre deosebire de algoritmul folosit la vectori, unde testul consta în verificarea indicilor (dacă au fost parcurși toți vectorii), în cazul fișierelor testul constă în verificarea sfârșitului de fișier.

```
#include <iostream.h>
#include <fstream.h>
void main()
{char nume1[20], nume2[20], nume3[20];
 int nr1, nr2;
 cout<<"nume fisier 1="; cin>>nume1;
 cout<<"nume fisier 2="; cin>>nume2;
 cout<<"nume fisier interclasat ="; cin>>nume3;
 fstream f1(nume1, ios::in), f2(nume2, ios::in),
           f3(nume3, ios::out);
 f1>>nr1; f2>>nr2;
 while (!f1.eof() && !f2.eof())
   if (nr1 <= nr2)
     {f3<<nr1<<endl; f1>>nr1;}
   else
     {f3<<nr2<<endl; f2>>nr2;}
   if (!f1.eof())
     while (!f1.eof())
       {f3<<nr1<<endl; f1>>nr1;}
   else
     while (!f2.eof())
       {f3<<nr2<<endl; f2>>nr2;}
 f1.close(); f2.close(); f3.close();
}
```

Enunțul problemei 5: Se scriu mai multe numere într-un fișier, câte un număr pe fiecare linie. Se citește un număr *n* de la tastatură. Să se afișeze din fișier toate numerele începând cu al *n*-lea număr. Afișarea numerelor se va face pe aceeași linie.

Se folosesc **identificatorii**:

- ✓ variabilele *nr* pentru citirea unui număr, *n* pentru poziția din care se face afișarea și *nume* pentru numele fișierului;
- ✓ *f1* și *f2* pentru fluxurile de date ale fișierului: *f1* pentru scrierea în fișier și *f2* pentru citirea din fișier.

Scrierea numerelor în fișier se face cu format (lățimea câmpului de 5 poziții) pentru a putea poziționa pointerul pe al *n*-lea număr. Deoarece pointerul se deplasează octet cu octet, al *n*-lea număr începe după octetul *nx6* (fiecare număr fiind scris pe o linie în fișier, pentru fiecare linie sunt necesari 6 octeți: 5 pentru număr și unul pentru ca-

racterul **sfârșit de linie** '\n'). Pentru poziționarea pe al n -lea număr se folosește funcția `seekp(6*n, ios::beg)`. Referința este **beg** - începutul fișierului, iar deplasarea este $6*n$.

```
#include <iostream.h>
#include <fstream.h>
#include <iomanip.h>
void main()
{char nume[20];
 int nr, n;
 cout<<"nume fisier ="; cin>>nume;
 cout<<"n= "; cin>>n;
 fstream f1(nume, ios::out);
 while (cin>>nr) f1<<setw(5)<<nr<<endl;
 f1.close();
 fstream f2(nume, ios::in);
 f2.seekp(6*n, ios::beg);
 while (f2>>nr) cout<<nr<<" ";
 f2.close();
}
```

Enunțul problemei 6: Într-un fișier se scriu mai multe numere de cel mult patru cifre fiecare. Să se afișeze maximul dintre aceste numere. Problema se va rezolva în două variante:

- numerele se scriu pe același rând în fișier.
- numerele se scriu fiecare pe câte un rând; în această variantă să se precizeze și al câtelea număr este maximul.

Se folosesc **identificatorii**:

- ✓ variabilele *nr* pentru citirea unui număr, *max* pentru valoarea maximă, *poz* pentru poziția maximului și *nume* pentru numele fișierului;
- ✓ *f1* și *f2* pentru fluxurile de date ale fișierului: *f1* pentru scrierea în fișier și *f2* pentru citirea din fișier.

Scrierea numerelor în fișier se face cu format (lățimea câmpului de 5 poziții) pentru a putea identifica un număr din șirul de caractere, atunci când sunt scrise pe un rând (identificarea făcându-se prin separarea lor cu spațiu) și pentru a calcula numărul de ordine al numărului maxim. Pentru calculul numărului de ordine se folosește funcția `tellp()`. Deoarece funcția furnizează numărul octetului de la care începe numărul maxim, numărul de ordine se calculează împărțind la 6.

Varianta_a

```
#include <iostream.h>
#include <fstream.h>
#include <iomanip.h>
void main()
{char nume[20];
 int nr, max;
 cout<<"nume fisier ="; cin>>nume;
 fstream f1(nume, ios::out);
 while (cin>>nr) f1<<setw(5)<<nr;
 f1.close();
 fstream f2(nume, ios::in);
```



```

f2>>nr; max=nr;
while (!f2.eof())
{f2>>nr; if (max<nr) max=nr;}
f2.close();
cout<<"maxim= "<<max;
}

```

Varianta_b

```

#include <iostream.h>
#include <fstream.h>
#include <iomanip.h>
void main()
{char nume[20];
 int nr, max;
 long poz;
 cout<<"nume fisier ="; cin>>nume;
 fstream f1(nume, ios::out);
 while (cin>>nr) f1<<setw(5)<<nr<<endl;
 f1.close();
 fstream f2(nume, ios::in);
 f2>>nr; max=nr;
 while (!f2.eof())
 {f2>>nr;
  if (max<nr) {max=nr; poz=f2.tellp()/6;} }
 f2.close();
 cout<<endl<<"maxim= "<<max<<" in pozitia "<<poz;
}

```

Enunțul problemei 7: Dintr-un fișier care conține mai multe numere întregi, scrise fiecare pe câte un rând, să se creeze două fișiere: unul cu numerele pare și altul cu numerele impare.

Se folosesc identificatorii:

- ✓ variabilele *nr* pentru citirea unui număr și variabilele *nume1* pentru numele fișierului sursă și *nume2* și *nume3* pentru numele fișierelor destinație;
- ✓ *f1*, *f2* și *f3* pentru fluxurile de date ale fișierelor: *f1* pentru fișierul sursă din care se citește, și *f2* și *f3* pentru fișierele destinație în care se scrie.

```

#include <iostream.h>
#include <fstream.h>
void main()
{char nume1[20], nume2[20], nume3[20];
 int nr;
 cout<<"nume fisier sursa="; cin>>nume1;
 cout<<"nume fisier numere pare="; cin>>nume2;
 cout<<"nume fisier numere impare ="; cin>>nume3;
 fstream f1(nume1, ios::in),
          f2(nume2, ios::out), f3(nume3, ios::out);
 f1>>nr;
 while (!f1.eof())
 {if (nr%2==0) f2<<nr<<endl;
  else f3<<nr<<endl;
  f1>>nr;}
 f1.close(); f2.close(); f3.close();
}

```

Evaluare

Răspundeți:

1. Descrieți cum sunt organizate datele în fișier. Dați exemple de probleme care, pentru a putea fi rezolvate, necesită organizarea datelor în fișiere.
2. Comparați cele două structuri de date: tabloul de memorie și fișierul.
3. Cum se declară un flux de date pentru un fișier deschis pentru consultare?
4. Găsiți greșeala din următoarea secvență de program:

```
fstream f("alfa.txt", ios::in);
while (cin >> ch) f << ch;
f.close();
```
5. Pentru operațiile din coloana A), alegeți din coloana B) constanta sau funcția care o realizează:

A	B
A1) deschide un fișier text pentru creare	B1) eoln()
A2) șterge înregistrările dintr-un fișier text	B2) app
A3) testează sfârșitul de linie într-un fișier text	B3) copy()
A4) copiază un fișier text într-un alt fișier text	B4) in
A5) testează sfârșitul unui fișier de text	B5) remove()
A6) deschide un fișier text pentru citire	B6) 'n'
A7) deschide un fișier text pentru adăugare	B7) eof()
	B8) out
	B9) noreplace
	B10) Ctrl+Z

Adevărat sau fals:

1. Numărul de înregistrări ale unui fișier se stabilește la crearea lui și nu mai poate fi modificat.
2. Fișierul este o structură de date omogenă creată în memoria externă.
3. Un fișier se termină întotdeauna cu caracterul eof.
4. Fișierul, fiind o structură de date liniară, ocupă o zonă continuă de memorie.
5. Un fișier text are înregistrări cu lungime fixă.
6. Un fișier text nu poate fi redenumit. Soluția pentru a-i schimba numele este de a-l copia sub un alt nume și de a șterge vechiul fișier.
7. Un fișier text poate conține orice caracter ASCII, mai puțin caracterele albe.
8. Pentru a crea un flux de date pentru un fișier text este suficient să se precizeze numele logic, numele fizic și modul de acces al fișierului.
9. Cu declarația următoare, se poate folosi fluxul de date *f* pentru crearea unui fișier:

```
fstream f("alfa.txt", ios::noreplace);
```

Alegeți:

1. Care dintre operațiile de mai jos nu se pot executa într-un fișier text:

- a) redenumirea
b) adăugarea de informații la sfârșit
c) inserarea unui caracter
d) ștergerea unei linii de text
2. Care dintre variantele de mai jos nu este un mod de acces care să poată fi precizat la deschiderea fișierului:
a) consultare
b) creare
c) adăugare
d) redenumire
3. Dacă fluxul de date f a fost declarat pentru un fișier care se creează, iar variabila ch este de tip caracter, și se execută instrucțiunea:

```
while (cin >> ch) f << ch;
```

și se citește de la tastatură (s-a notat spațiul cu b):

25 b 75.5 enter a b enter Ctrl+Z

se va afișa:

- a) 25 b 75.5
a b
b) 2575ab
c) 25 b 75.5
a b
d) 2575.5ab
4. Dacă fluxul de date f a fost declarat pentru un fișier care se consultă și variabila ch este de tip caracter și se execută instrucțiunea:

```
while (f >> ch) cout << ch;
```

și conținutul fișierului este (s-a notat spațiul cu b):

25 b 75.5 enter a b enter Ctrl+Z

fișierul va conține:

- a) 25 b 75.5
a b
b) 2575ab
c) 2575.5
ab
d) 2575.5ab

Rezolvați:

- Fișierele text *alfa.txt* și *beta.txt* conțin numele unor persoane, câte o linie pentru fiecare persoană. Știind că în fiecare fișier numele sunt memorate în ordine alfabetică, scrieți un program care să construiască fișierul *gama.txt* care să conțină toate numele din cele două fișiere date, în ordinea alfabetică.
- Scrieți un program care generează toate numerele prime strict mai mici decât n (n număr natural). Valoarea variabilei n se citește de la tastatură. Numerele prime generate vor fi scrise în fișierul text *prime.txt*, câte unul pe linie.
- Pe prima linie a fișierului text *alfa.txt*, se găsește o succesiune de cel puțin două și cel mult 200 de caractere, caractere care pot fi doar litere mici. Scrieți un program care citește de la tastatură un număr natural n ($0 < n < 100$) și stabilește dacă există în fișier vreo literă ce apare de exact n ori. Programul afișează pe ecran mesajul "da" în cazul în care există cel puțin o literă cu proprietatea menționată și mesajul "nu" în caz contrar.
- Fișierul text *litere.txt* conține mai multe cuvinte scrise cu litere mici, câte un cuvânt pe fiecare linie. Scrieți un program care să afișeze litera care apare de cele mai multe ori. Dacă există mai multe astfel de litere, se vor afișa toate.

5. Scrieți un program care citește de la tastatură un număr natural n ($0 < n \leq 15$) și o literă c și creează un fișier text cu numele *caracter.txt* ce conține pe prima linie un caracter c , pe a doua linie două caractere c nedespărțite prin spații, pe linia a treia trei caractere c nedespărțite prin spații etc. Ultima linie a fișierului trebuie să fie linia a n -a care conține n caractere c nedespărțite prin spații. Afișați conținutul fișierului.
6. În fișierul text *numere.txt* se află mai multe numere naturale, de câte cel mult patru cifre fiecare, scrise pe un singur rând. Scrieți un program care afișează pe ecran câte valori distincte există în fișier.
7. În fișierul text *alfa.txt* se află mai multe numere naturale, de cel mult trei cifre fiecare, scrise pe un singur rând. Scrieți un program care creează un alt fișier text *beta.txt* care să conțină exact aceleași numere din fișierul *alfa.txt*, câte unul pe linie, în ordinea crescătoare a valorilor acestora. (Indicație. Sortarea numerelor se va face într-un vector).
8. În fișierul text *numere.txt* se află mai multe numere naturale din intervalul $[0, 10000]$, scrise pe un singur rând. Să se creeze fișierul *pare.txt* care să conțină, câte una pe linie, doar acele valori, din fișierul *numere.TXT*, care au suma cifrelor un număr par.
9. În fișierul text *alfa.txt* se află, câte unul pe linie, mai multe numere. În fișierul text *divizori.txt* se vor scrie pe câte o linie, în aceeași ordine, divizorii unui număr din fișierul *alfa.txt*. Afișați pe ecran, pe câte un rând, folosind informațiile din cele două fișiere: *Numărul ... are divizorii*
10. Să se verifice dacă două fișiere text conțin același număr de linii. Dacă au același număr de linii să se afișeze mesajul "Număr egal de linii", altfel să se afișeze un mesaj prin care să se precizeze care fișier are mai multe linii.
11. Se citesc de la tastatură trei numere întregi n , a și b și un șir de n numere reale care se scriu într-un fișier text *alfa.txt* toate pe același rând. Să se afișeze câte dintre numerele din fișier se află în afara intervalului $[a, b]$.
12. Un fișier *alfa.txt* conține mai multe cuvinte, câte unul pe fiecare rând. Se citește de la tastatură un caracter c . Afișați numărul de înregistrări ale fișierului care conțin cuvinte care încep cu caracterul c .
13. Se citesc de la tastatură mai multe triplete de numere întregi (a, b, c) care reprezintă laturile unui triunghi și se scriu într-un fișier *alfa.txt*, câte o tripletă pe fiecare rând. Se citesc apoi din fișier aceste triplete de numere și se analizează tipul triunghiului (oarecare, echilateral, isoscel, dreptunghic, dreptunghic isoscel) și se scrie tipul triunghiului pe un rând, în fișierul *beta.txt*. Fișierul *beta.txt* va avea tot atâtea înregistrări ca și fișierul *alfa.txt*. Afișați apoi pe ecran, pe câte un rând, informații despre fiecare triunghi: dimensiunile laturilor și tipul triunghiului.
14. Se citesc de la tastatură mai multe triplete de numere întregi (a, b, c) care se scriu într-un fișier *alfa.txt*, câte o tripletă pe fiecare rând. Se citesc apoi din fișier aceste triplete de numere și se analizează dacă ele reprezintă laturile unui triunghi și se scrie pe un rând, în fișierul *beta.txt*, aria triunghiului, dacă ele reprezintă laturile unui triunghi, și 0 dacă nu reprezintă laturile unui triunghi. Afișați apoi pe ecran, pe câte un rând, informații despre triunghiurile găsite: dimensiunile laturilor și aria.

Anexa 1

Mediul de programare C++

Pentru a obține un program executabil, trebuie parcurse următoarele etape:

- ✓ editarea programului sursă;
- ✓ compilarea programului sursă;
- ✓ editarea legăturilor programului obiect;
- ✓ lansarea în execuție a programului executabil;
- ✓ depanarea interactivă a programului;
- ✓ memorarea programului sursă și a programului executabil, pe un suport de informație.

Toate aceste operații pot fi asigurate de diferite programe: editor de texte, compilator, editor de legături, program depanator. Pentru a ușura munca programatorului, aceste programe pot fi încorporate într-un mediu integrat care să asigure prin intermediul aceleiași interfețe toate aceste operații. Acest mediu integrat se numește **mediu de programare**.

Mediul de programare C++ asigură următoarele operații:

- ✓ editarea programului sursă;
- ✓ administrarea fișierelor sursă: crearea, salvarea, deschiderea, închiderea, tipărirea;
- ✓ compilarea programului sursă;
- ✓ editarea legăturilor și obținerea programului executabil;
- ✓ lansarea în execuție a programului;
- ✓ depanarea și corectarea erorilor;
- ✓ configurarea mediului de programare;
- ✓ salvarea pe disc a programului executabil;
- ✓ autodocumentarea prin sistemul de asistență (**Help**);
- ✓ istoricul opțiunilor din casetele de dialog;
- ✓ administrarea mai multor ferestre.

Atunci când într-o casetă de text dintr-o casetă de dialog scrieți o valoare a parametrului (o cale de director, un șablon de fișiere etc.) ea se păstrează într-un istoric și vă permite să o alegeți, prin intermediul unei liste ascunse, la următoarea deschidere a casetei de dialog. Aceasta este facilitatea de istoric al opțiunilor din casetele de dialog.

Mediul de programare C++ vă oferă instrumente pentru depanarea următoarelor tipuri de erori:

- ✓ erori sintactice (în etapa de compilare);
- ✓ erori semantice (în etapa de execuție);
- ✓ erori logice sau de concepție (în etapa de testare).

Mediul de programare C++ recunoaște următoarele extensii de fișiere:

- ✓ **.exe** – program executabil;
- ✓ ***.cpp** sau ***.c** – program sursă;
- ✓ ***.obj** – program obiect;
- ✓ ***.lib** – bibliotecă C++;
- ✓ ***.h** – fișier antet;
- ✓ ***.bak** – versiunea anterioară a programului sursă.

O sesiune de lucru C++ durează din momentul în care lansați în execuție mediul de programare (aplicația **Bc** sau **Bcw**) și până în momentul în care abandonați mediul de lucru cu

Informații

opțiunea c
interfața m
este reali
grafic (apl
regulă 24
un caracte

Interfața

Interfața m

- ✓ **Bara c**
niurilor
- ✓ **Bara c**
grafice
- ✓ **Spațiu**
- ✓ **Bara c**
pentru
select

Bara cu t

- ✓ **File** –
creare
- ✓ **Edit** –
- ✓ **Search**
folosita
- ✓ **Run** –
- ✓ **Comp**
putând
- ✓ **Debug**
- ✓ **Proje**
- ✓ **Option**
editare
comun
- ✓ **Wind**
- ✓ **Help** –

Pentru ad
niu apăsa

La un mo
conțin tex
exemplu,
nistrarea
le obținut
screen...
și Registe

Pentru rec
scurtătura

¹ Fișierul pr
vă ajută să

opțiunea de meniu **Exit** **File** sau cu scurtătura **Alt+X**. În timpul sesiunii de lucru este afișată interfața mediului de programare **C++**. Această interfață folosește un sistem de meniuri și este realizată, în funcție de mediul de programare, în mod text (aplicația **Bc**) sau în mod grafic (aplicația **Bcw**). În modul text, ecranul calculatorului este împărțit în linii și coloane (de regulă 24 sau 25 de linii și 80 de coloane); la intersecția unei linii cu o coloană este generat un caracter.

Interfața mediului de programare

Interfața mediului de programare **C++** conține următoarele elemente:

- ✓ **Bara cu titluri de meniuri** este afișată pe prima linie a ecranului. Ea conține titlurile meniurilor disponibile în mediul de programare.
- ✓ **Bara de instrumente** este afișată sub bara cu titluri de meniuri numai în cazul interfeței grafice. Ea conține pictograme scurtături pentru opțiunile de meniu mai des folosite.
- ✓ **Spațiul de lucru** este zona în care pot fi deschise mai multe ferestre.
- ✓ **Bara de informare** este afișată în partea inferioară a ecranului și conține fie scurtături pentru operațiile din fereastra activă, fie informații despre titlul sau opțiunea de meniu selectată.

Bara cu titluri de meniuri afișează următoarele titluri de meniuri:

- ✓ **File** – conține opțiuni pentru administrarea fișierelor cu programe sursă (deschidere, creare, salvare, tipărire);
- ✓ **Edit** – conține opțiuni pentru editarea programului sursă;
- ✓ **Search** – conține opțiuni pentru căutarea șirurilor de caractere sau a erorilor, putând fi folosite în editarea programului sursă sau în depanarea lui;
- ✓ **Run** – conține opțiuni pentru executarea programului;
- ✓ **Compile** – conține opțiuni pentru compilarea programului sursă, rezultatul compilării putând fi depozitat în memoria internă sau într-un fișier obiect pe disc;
- ✓ **Debug** – conține opțiuni care vă ajută în depanarea programului;
- ✓ **Project** – conține opțiuni care vă ajută la realizarea proiectelor¹;
- ✓ **Options** – conține opțiuni pentru configurarea mediului de programare: funcțiile de editare, compilare, editare de legături și depanare, cât și echipamentele care asigură comunicarea cu mediul de programare (mouse și monitor);
- ✓ **Window** – conține opțiuni pentru administrarea ferestrelor;
- ✓ **Help** – conține opțiuni pentru autodocumentare.

Pentru activarea barei cu titluri de meniuri, apăsați tasta **F10**. Pentru a deschide direct un meniu apăsați tastele **Alt+<litera_de_identificare>** (de exemplu, **Alt+E** pentru meniul **Edit**).

La un moment dat, pe ecran pot fi deschise mai multe ferestre: atât ferestre de editare care conțin textul programelor sursă, cât și ferestre care vă ajută să depanați programul (de exemplu, în aplicația **Bc** ferestrele **Clipboard**, **Watch**, **Register** sau **Output**). Pentru administrarea ferestrelor, veți folosi opțiunile din meniul **Window**. Astfel, în aplicația **Bc** rezultatele obținute în urma executării programului le puteți vedea pe tot ecranul cu opțiunea **User screen...** (scurtătura **Alt+F5**) sau într-o fereastră, cu opțiunea **Output**, iar ferestrele **Watch** și **Register** pot fi deschise și ele cu opțiunea corespunzătoare din meniul **Window**.

Pentru redimensionarea sau mutarea unei ferestre, alegeți opțiunea **Size/Move** sau apăsați scurtătura **Ctrl+F5**. Bara de informare va afișa tastele pe care le puteți utiliza pentru aceste

¹ Fișierul proiect se creează atunci când, pentru rezolvarea unei probleme, creați mai multe fișiere sursă. El vă ajută să țineți mai ușor evidența colecției de fișiere sursă necesare pentru rezolvarea problemei.

operații: folosind tastele cu săgeți veți muta fereastra în sensul săgeții, folosind tastele cu săgeți împreună cu tasta **Shift** veți redimensiona fereastra pe direcția săgeții, apăsând tasta **Enter** veți fixa noua poziție sau dimensiune a ferestrei, iar apăsând tasta **Esc** veți reveni la vechea poziție sau dimensiune a ferestrei.

Opțiunea **Zoom** și scurtătura **F5** au efect de comutator, producând maximizarea ferestrei sau refacerea ei la dimensiunea anterioară.

Pentru aranjarea ferestrelor în cascadă sau în mozaic alegeți opțiunea **Cascade**, respectiv **Tile**. Pentru activarea unei ferestre, procedați astfel: pentru aplicația **Bcw** fie executați clic pe fereastră, fie alegeți numele ferestrei din lista cu ferestre deschise în meniul **Window**, iar pentru aplicația **Bc** fie alegeți opțiunea **Next** sau apăsați scurtătura **F6** pentru fereastra următoare, fie alegeți opțiunea **Previous** sau scurtătura **Shift+F6** pentru fereastra precedentă, fie opțiunea **List...Window** sau scurtătura **Alt+O** care deschide caseta de dialog **Windows list** în care este afișată lista ferestrelor deschise și din care puteți alege fereastra pe care vreți să o activați.

Pentru închiderea ferestrei active alegeți opțiunea **Close** sau apăsați scurtătura **Alt+F3** sau executați clic pe butonul de închidere al ferestrei.

Editarea programului sursă

Cu ajutorul editorului de texte încorporat (program specializat în scrierea textelor în format ASCII) se scrie programul de la tastatură în memoria internă și apoi se salvează pe un suport de informație. Operația se numește **editarea programului**, iar programul obținut este **programul sursă**.

Fereastra de editare (fereastra în care se scrie programul sursă) conține pe primul rând o bară de titlu, iar pe ultimul rând coordonatele cursorului în cadrul textului și bara de derulare pe orizontală. Coordonatele cursorului sunt specificate sub forma **x:y**, unde **x** este numărul liniei, iar **y** numărul coloanei. În bara de titlu a ferestrei sunt afișate: în extremitatea stângă butonul de închidere a ferestrei [■], care poate fi acționat doar cu mouse-ul, în centru titlul ferestrei, iar în extremitatea dreaptă numărul de ordine al ferestrei. Titlul ferestrei va fi numele fișierului în care este salvat programul sursă. Dacă nu l-ați salvat încă într-un fișier pe disc, titlul ferestrei va fi **NONAMExx.CPP** (**xx** este un număr care se atribuie, începând cu 00, ferestrelor de editare deschise al căror conținut nu a fost salvat).

Prin intermediul opțiunilor din meniul **File** puteți să executați următoarele operații:

1. **Crearea unui fișier.** Pentru a deschide o fereastră de editare în care să scrieți textul unui nou program sursă alegeți opțiunea **New**.
2. **Deschiderea fișierului.** Pentru deschiderea unui fișier care conține un program sursă alegeți opțiunea **Open...** sau scurtătura **F3**. Se va deschide caseta de dialog **Open a File**. În caseta de text **Name** scrieți calea de director și eventual șablonul fișierelor care vreți să fie afișate și, apoi, din lista **Files**, alegeți numele fișierului pe care vreți să-l deschideți. Dacă fișierul a mai fost deschis în sesiunea de lucru curentă, îl mai puteți deschide alegându-i numele din lista ascunsă **Name**, care conține istoricul fișierelor deschise. În partea inferioară a casetei de dialog există o zonă cu informații despre fișierul sau directorul selectat în lista **Files** (spațiul ocupat pe disc – în octeți –, data și ora ultimei actualizări). Dacă vreți să deschideți fișierul într-o nouă fereastră de editare, acționați declanșatorul **Open**, iar dacă vreți să-l deschideți în fereastra de editare activă, acționați declanșatorul **Replace**. Dacă doriți informații despre controalele din caseta de dialog, acționați declanșatorul **Help**.

3. **Salva**
trebu
prin
unita
Open
care
să sa
vreți s
4. **Tipă**
de ed
5. **Schi**
dir...
Direc
tree
schir
apăsa
direct

Pentru ed
ducerea te
lucru este

- Pentru dep
- ✓ tastele
 - ✓ **Ctrl+A**
 - ✓ **Ctrl+E**
 - ✓ **Home**
 - ✓ **PageUp**
 - ✓ **Ctrl+Q**
 - ✓ **Ctrl+Q**
 - ✓ **Ctrl+Q**
 - selecta
 - ✓ **Ctrl+P**

- Pentru șle
- ✓ **Delete**
 - ✓ **Backsp**
 - spațiu
 - ✓ **Ctrl+F**
 - ✓ **Ctrl+Q**
 - ✓ **Ctrl+Y**

Puteți să l
editorului
acel cuvâ
tastele **Ctrl**
memorie **C**

² **Ctrl+Q,E**
E sau **X**. Da
pe ultima lin

1. **Salvarea programului sursă într-un fișier pe disc.** Dacă este un program nou va trebui să-l salvați cu opțiunea **Save as...** care deschide caseta de dialog **Save File As** prin intermediul căreia comunicați numele fișierului și locul în care va fi salvat (nume unitate de disc și cale de director). Caseta conține aceleași obiecte ca și caseta de dialog **Open a File**, mai puțin declanșatorul **Replace**. Veți alege această opțiune și în cazul în care doriți să salvați modificările făcute într-un program sursă, într-un alt fișier. Dacă vreți să salvați modificările în același fișier, alegeți opțiunea **Save** sau scurtătura **F2**. Dacă vreți să salvați toate modificările din ferestrele de editare, alegeți opțiunea **Save all**.
2. **Tipărirea programului sursă.** Pentru tipărirea programului sursă, se activează fereastra de editare în care se găsește textul programului și se alege opțiunea **Print**.
3. **Schimbarea directorului curent** (numai în aplicația **Bc**). Se face cu opțiunea **Change dir...** care deschide caseta de dialog **Change Directory**. Scrieți în caseta de text **Directory name** sau alegeți din lista ascunsă numele unității de disc. În zona **Directory tree** este afișată structura arborescentă care se dezvoltă din directorul curent. Veți schimba directorul curent direct pe arbore: poziționați cursorul pe numele directorului și apăsați tasta **Enter**, după care acționați declanșatorul **Chdir**. Dacă vreți să reveniți la directorul inițial, acționați declanșatorul **Revert**.

Pentru editarea programului sursă veți folosi opțiunile din meniurile **Edit** și **Search**. Introducerea textului de la tastatură se poate face fie cu inserare, fie cu suprascriere. Acest mod de lucru este controlat de tasta **Insert** sau cu tastele **Ctrl+V** care au efect de comutator.

Pentru deplasarea cursorului puteți folosi următoarele taste:

- tastele cu săgeți – o poziție în sensul săgeții;
- Ctrl+A/Ctrl+F** – la începutul cuvântului anterior/următor;
- Ctrl+E/Ctrl+X** – prima/ultima linie de pe ecran;
- Home/End** – începutul/sfârșitul liniei curente;
- PageUp/PageDown** – pagina anterioară/următoare;
- Ctrl+Q,E/X²** sau **Ctrl+Home/Ctrl+End** – începutul/sfârșitul ecranului;
- Ctrl+Q,R/C** sau **Ctrl+PageUp/Ctrl+PageDown** – începutul/sfârșitul textului;
- Ctrl+Q,B/K** – începutul/sfârșitul blocului de text selectat (au efect și dacă s-a anulat selectarea blocului);
- Ctrl+P** – poziția anterioară a cursorului.

Pentru ștergere puteți folosi următoarele taste:

- Delete** – caracterul din poziția cursorului (nu șterge un bloc de text selectat);
- Backspace** – caracterul din stânga cursorului (dacă la stânga cursorului sunt mai multe spații, vor fi șterse toate spațiile până la nivelul de indentare anterior);
- Ctrl+T** – șterge toate caracterele din poziția cursorului până la sfârșitul cuvântului;
- Ctrl+Q,Y** – șterge caracterele de la dreapta cursorului până la sfârșitul rândului;
- Ctrl+Y** – șterge rândul în care se găsește cursorul.

Puteți să lucrați cu blocurile de texte. Le veți selecta folosind aceeași metodă ca și în cazul editorului de texte **Notepad**. În plus, puteți selecta un cuvânt astfel: poziționați cursorul pe acel cuvânt și apăsați tastele **Ctrl+K,T**. Dacă vreți să anulați selectarea unui bloc, apăsați tastele **Ctrl+K,H**. Manevrarea blocului selectat se poate face prin intermediul zonei de memorie **Clipboard**. În meniul **Edit**, aveți următoarele opțiuni:

Ctrl+Q,E înseamnă că mai întâi se apasă tastele **Ctrl+Q**, după care se apasă tasta **E**. **E/X** înseamnă **E** sau **X**. Dacă se apasă **E**, cursorul se deplasează pe prima linie, iar dacă se apasă **X**, se deplasează pe ultima linie.

- ✓ **Cuț** (scurtătura **Shift+Delete**) mută în **Clipboard** blocul selectat.
- ✓ **Copy** (scurtătura **Ctrl+Insert**) copiază în **Clipboard** blocul selectat. **Paste** (scurtătura **Shift+Insert**) inserează în poziția cursorului blocul din **Clipboard**.
- ✓ **Show Clipboard** (numai în aplicația **Bc**) deschide o fereastră în care puteți să vedeți conținutul zonei de memorie **Clipboard**. În această fereastră veți vedea toate blocurile de texte care au fost transferate prin **Clipboard** în sesiunea de lucru curentă. Ultimul bloc transferat va fi evidențiat. Puteți să manevrați textul din **Clipboard** ca și cum ar fi un text dintr-o fereastră de editare.
- ✓ **Clear** (scurtătura **Ctrl+Delete**) șterge blocul de text selectat.

Puteți să mutați sau să copiați un bloc fără să folosiți zona de memorie **Clipboard**, astfel: selectați blocul, mutați cursorul în poziția în care vreți să-l copiați sau să-l mutați și apăsați tastele **Ctrl+K,V** pentru mutare, respectiv **Ctrl+K,C** pentru copiere.

Cu blocul selectat mai puteți executa următoarele operații:

- ✓ **Tipăriți la imprimantă blocul selectat** dacă apăsați tastele **Ctrl+K,P**.
- ✓ **Salvați blocul selectat, într-un fișier pe disc**, apăsând tastele **Ctrl+K,W**. Se deschide caseta de dialog **Write Block to File** prin intermediul căreia veți specifica un nume pentru fișier. Dacă nu precizați extensia, i se va atribui automat extensia **.cpp**.
- ✓ **Inserați în poziția cursorului un bloc de text preluat dintr-un fișier** de pe disc, apăsând tastele **Ctrl+K,R**. Se deschide caseta de dialog **Read Block from File** prin intermediul căreia veți specifica numele fișierului.
- ✓ **Deplasați blocul** cu o coloană la dreapta sau la stânga dacă apăsați tastele **Ctrl+K,I**, respectiv **Ctrl+K,U**.

Dacă vreți să anulați ultimele operații de editare, folosiți opțiunea de meniu **Undo** **Edit** (scurtătura **Alt+BackSpace**) sau puteți să refaceți acțiunile anulate, cu opțiunea de meniu **Redo** **Edit** (scurtătura **Alt+BackSpace**).

Puteți să inserați în programul sursă marcare (maxim 10) care să vă permită poziționarea rapidă a cursorului pe o anumită instrucțiune. Poziționați cursorul pe instrucțiunea la care vreți să fixați marcajul și apăsați tastele **Ctrl+K,<n>** ($n=0, 1, 2, \dots, 9$). Pentru localizarea marcajului inserat apăsați tastele **Ctrl+Q,<n>** ($n=0, 1, 2, \dots, 9$).

În meniul **Search** aveți mai multe opțiuni pentru operații de căutare și înlocuire:

- ✓ **Find...** vă permite să căutați un șir de caractere (scurtătura **Ctrl+Q,F**). Prin intermediul casetei de dialog **Find** puteți să comunicați șirul de caractere pe care îl căutați (în caseta de text **Text to find**) și modul în care se face căutarea. Din grupul de butoane radio **Direction** alegeți direcția de căutare față de poziția cursorului: **Forward** – înainte sau **Backward** – înapoi. Din grupul de butoane radio **Scope** alegeți domeniul de căutare: **Global** – în întreg programul sursă sau **Selected text** – în blocul de text selectat din fereastra de editare. Din grupul de butoane radio **Origin** alegeți poziția din care începe căutarea: **From cursor** – din poziția cursorului, sau **Entire scope** – de la începutul domeniului de căutare. Prin intermediul comutatoarelor din zona **Options** puteți stabili ce căutați. Dacă activați comutatorul **Case sensitive**, se va face diferență între literele mari și mici, iar dacă activați comutatorul **Whole words only**, șirul de caractere căutat va fi considerat un cuvânt.
- ✓ **Replace** vă permite căutarea unui șir de caractere și înlocuirea lui cu un alt șir de caractere (scurtătura **Ctrl+Q,A**). Caseta de dialog **Replace** conține în plus, față de caseta de dialog **Find**, caseta de text **New text** în care scrieți șirul de caractere cu care se face înlocuirea, și declanșatorul **Change all** pe care îl acționați dacă vreți să se facă înlocuirea tuturor șirurilor de caractere găsite.

- ✓ **Search again** repetă ultima operație de căutare sau înlocuire (scurtătura **Ctrl+L**).
- ✓ **Go to line number...** mută cursorul în linia specificată prin intermediul casetei de dialog **Go to Line Number**.
- ✓ **Locate Function...** caută o declarație de funcție. Este utilă atunci când depanați programul.

Operația de editare poate fi controlată prin intermediul obiectelor din caseta de dialog **Editor Options** pe care o deschideți cu opțiunea **Editor... ∈ Environment ▸ ∈ Options**. În caseta de text **Tab size** scrieți numărul de coloane peste care vreți să se sară când apăsați tasta **Tab** (implicit, 8 coloane). Activați comutatorul:

- ✓ **Create backup files** – dacă vreți să se creeze o copie cu versiunea anterioară a programului sursă, într-un fișier cu extensia **.bak**;
- ✓ **Autoindent mode** – dacă vreți să fie activat modul de indentare automată prin care, la apăsarea tastei **Enter**, cursorul se va poziționa sub primul caracter de pe rândul anterior;
- ✓ **Backspace unindents** – dacă vreți ca, prin apăsarea tastei **Backspace**, să se șteargă toate spațiile până la nivelul de indentare anterior.



Editorul de texte folosește **marcaje cu culoare** pentru a identifica mai ușor elementele folosite în cadrul unui program: cuvinte cheie, identificatori, constante (pentru fiecare tip de constantă poate fi folosită altă culoare), comentarii, caractere ilegale, directive de pre-compilare. Puteți modifica marcajele de culoare predefinite, prin intermediul controalelor din caseta de dialog **Color** (aplicația **Bc**) sau **Highlighting** (aplicația **Bcw**) pe care o deschideți cu opțiunea de meniu **Color... /Highlight... ∈ Environment ▸ ∈ Options**.

Traducerea programului

În această fază fiecare instrucțiune din programul sursă este tradusă într-o secvență de instrucțiuni în cod mașină care pot fi executate de calculator, obținându-se **modulele obiect**. Operația se execută sub controlul unui program numit **compilator**. Operația se numește **compilare** și programul obținut se numește **program obiect**.

Dacă programul compilator detectează cel puțin o **eroare sintactică** (un cuvânt greșit, un separator lipsă, o variabilă de memorie care nu a fost declarată, apelarea unei funcții cu un număr greșit de parametri etc.), el va afișa **mesaje de eroare** într-o fereastră cu mesaje (fereastră **Message**), iar cursorul va fi poziționat în programul sursă pe prima linie în care s-a detectat o eroare. Executând clic pe o eroare din fereastră **Message**, în programul sursă, cursorul va fi poziționat pe linia în care a fost detectată eroarea. Dacă linia indicată este corectă, înseamnă că eroarea este pe linia precedentă (nu ați scris separatorul de sfârșit de instrucțiune ; sau o paranteză acoladă lipsește sau este în plus). În acest caz, autorul programului poate să modifice fișierul sursă folosind programul editor, după care va compila din nou programul. Operațiile de modificare cu editorul și de testare cu compilatorul, se vor executa până când compilatorul nu va mai detecta erori. În timpul compilării pot fi afișate și **mesaje de avertizare**. Aceasta înseamnă că din punct de vedere al compilatorului construcția are sintaxa corectă, dar este posibil ca în contextul programului să fie totuși o greșeală (de exemplu, ați declarat o variabilă de memorie pe care nu ați folosit-o în program, ați folosit o variabilă de memorie fără să-i atribuiți o valoare, ați folosit în expresia instrucțiunilor **if**, **while** sau **do...while** operatorul de atribuire = în locul operatorului relațional == etc.). Este bine să nu ignorați aceste avertismente, deoarece ele vă ajută să depistați și să corectați încă din faza de compilare erori de semantică sau de logică.

Atunci când compilatorul nu mai găsește erori în programul sursă, înseamnă că traducerea s-a făcut corect și rezultatul traducerii va fi salvat într-un fișier obiect. Programul obiect obținut nu este un program executabil deoarece modulele obiect sunt asemănătoare pieselor de „puzzle”: sunt fragmente care necesită să fie asamblate pentru a forma o imagine unitară, adică programul executabil.

Pentru compilare puteți folosi una dintre următoarele opțiuni din meniul **Compile**:

- ✓ **Compile** (scurtătura **Alt+F9**) – compilează programul sursă;
- ✓ **Remove messages** (scurtătura **Alt+F9**) – recompilează programul sursă și toate bibliotecile folosite de program.

Editarea legăturilor

Modulele obiect sunt legate unele de altele astfel încât să se obțină un **program executabil**. Operația se numește **editare de legături** (*link edit*) și este executată de către un program numit **editor de legături** (*linkage editor*). Pentru a obține programul executabil, pot fi legate și module obiect care există deja în bibliotecile³ sistemului. Nu este necesar să se execute separat operația de editare a legăturilor, deoarece ea va fi făcută automat atunci când se cere executarea programului sursă. Dacă vreți să executați această operație separat, puteți folosi următoarele opțiuni din meniul **Compile**:

- ✓ **Make** (scurtătura **F9**) – compilează programul sursă și editează legăturile cu alte module obiect;
- ✓ **Link** (scurtătura **Alt+F9**) – recompilează programul sursă și editează legăturile.
- ✓ **Build All** (scurtătura **Alt+F9**) – recompilează toate fișierele sursă ale unui proiect și editează legăturile.

Încărcarea și lansarea în execuție

Programul sursă poate fi lansat în execuție pentru a produce rezultatele pentru care a fost creat. În timpul executării programului pot să apară **erori semantice** (încercarea de a deschide un fișier care nu există, împărțirea la 0 etc.). Aceste erori vor duce la oprirea execuției programului. În acest caz, autorul programului va depista eroarea și va modifica fișierul sursă cu programul editor, după care va compila din nou programul.

Pentru a lansa în execuție programul sursă din fereastra curentă, alegeți opțiunea **Run** sau folosiți scurtătura **Ctrl+F9**. Se va crea **programul executabil**. Dacă în program există o instrucțiune care ciclează la infinit, pentru a opri execuția programului, se apasă tastele **Ctrl+Break(Pause)**. Instrucțiunea pe care s-a oprit execuția programului va fi evidențiată. Dacă apăsați scurtătura **Ctrl+F9**, se va relua execuția programului de la instrucțiunea la care a fost întrerupt și va cicla din nou la infinit. Pentru a relua execuția de la începutul programului, cu noi date de intrare, veți folosi opțiunea **Program reset** **Run** (scurtătura **Ctrl+F2**) care eliberează zona de memorie folosită de programul în execuție.

Pentru corectarea erorilor semantice semnalate în timpul executării programului, se comută în editorul de texte și, cu opțiunea **Find error** **Search**, se găsește locul erorii. Mai puteți folosi opțiunile de meniu **Previous Error** **Search** (scurtătura **Alt+F7**), pentru poziționare pe eroarea precedentă, și **Next Error** **Search** (scurtătura **Alt+F8**), pentru poziționare pe eroarea următoare.

³ **Biblioteca** (*library*) este un ansamblu de funcții executabile (secvențe de coduri binare executabile) stocate într-un fișier, care pot fi apelate în cadrul unui program.

Testarea și depanarea programului

Programul executabil poate fi testat. Testarea se face prin executarea repetată a programului, cu un **set complet de date de intrare**. În timpul testării se poate afla dacă există **erori de concepție** sau **erori logice** (programul nu lucrează corect, adică nu produce rezultatele dorite pentru fiecare set de date de intrare, sau ciclează la infinit) sau dacă rezultatele obținute nu au aspectul grafic dorit. Pentru remedierea acestor erori trebuie corectat programul sursă, cu ajutorul editorului, compilat din nou cu ajutorul compilatorului, editate legăturile și reluat testul. Operația de corectare a erorilor se numește **depanarea programului**. Ea se poate desfășura sub controlul unui program specializat numit **depanator de programe** (*debugger*), care ajută la detectarea instrucțiunii eronate.

Erorile logice sunt cel mai greu de detectat. Pentru detectarea lor veți putea folosi opțiunile din meniul **Debug** și **Run**, prin intermediul cărora puteți:

- ✓ să executați pas cu pas instrucțiunile din program,
- ✓ să urmăriți variabilele de memorie și expresiile în timpul execuției programului,
- ✓ să creați și să administrați **punctele de întrerupere** (*breakpoints*).

1. **Executarea pas cu pas a programului.** Executarea pas cu pas a programului înseamnă executarea pe rând, la cerere, a fiecărei instrucțiuni din program. În acest mod puteți să vedeți ordinea în care se execută instrucțiunile. Pentru executarea pas cu pas a programului puteți folosi opțiunile din meniul **Run**: opțiunea **Trace into** (scurtătura **F7**), care execută instrucțiunea următoare, chiar dacă ea face parte dintr-o funcție, și opțiunea **Step over** (scurtătura **F8**) care execută instrucțiunea următoare din program la sursă. Se poate alterna folosirea acestor opțiuni, în funcție de ceea ce doriți să vedeți la un moment dat: numai modul în care se execută fișierul sursă sau și modul în care se execută o anumită funcție definită în fișierul sursă.
2. **Urmărirea variabilelor de memorie și a expresiilor.** În timpul execuției programului este util să vizualizați conținutul variabilelor de memorie și valorile expresiilor. Aceste valori vor fi reevaluate după fiecare instrucțiune executată. Ele pot fi urmărite prin intermediul ferestrei **Watch** (de urmărire) pe care o deschideți cu opțiunea **Watch Window**. Pentru administrarea expresiilor din fereastra **Watch** puteți folosi opțiunile din submeniul **Watches** » **Debug**, și anume: **Add watch...** (scurtătura **Ctrl+F7**) care permite adăugarea la listă a noi expresii, **Delete watch** care înlătură din listă expresia selectată, **Edit watch...** care permite modificarea expresiei selectate, **Remove all watch** care înlătură toate expresiile din listă. Prin intermediul opțiunii de meniu **Evaluate/Modify...** **Debug** (scurtătura **Ctrl+F4**) puteți evalua și modifica variabile și expresii. Se deschide caseta de dialog **Evaluate and Modify** în care veți introduce, în caseta de text **Expression**, expresia care trebuie evaluată. În zona **Result** este afișat rezultatul evaluării, iar în caseta **New Value** introduceți noua valoare a expresiei. Dacă vreți numai să evaluați expresia, acționați declanșatorul **Evaluate**, iar dacă vreți să o și modificați, acționați declanșatorul **Modify**.
3. **Crearea punctelor de întrerupere.** Punctele de întrerupere sunt marcaje, în interiorul unui program, la care se oprește temporar execuția programului, astfel încât să puteți examina starea programului și conținutul variabilelor de memorie. Este mult mai avantajos să folosiți punctele de întrerupere decât să executați pas cu pas programul, deoarece veți face economie de timp în execuția programului pe care îl depanați. Administrarea punctelor de întrerupere se face prin intermediul opțiunilor din meniul **Debug**. Pentru a stabili un punct de întrerupere procedați astfel: poziționați cursorul pe instrucțiunea la care doriți să se întrerupă execuția programului (instrucțiunea care bănuieți că este cauza

erorii) și alegeți opțiunea **Toggle breakpoint** **Debug** (scurtătura **Ctrl+F8**). Operația se reia pentru fiecare punct de întrerupere pe care vreți să-l inserați în program. Dacă vreți să ștergeți un punct de întrerupere, poziționați cursorul pe linia la care a fost inserat și alegeți opțiunea **Toggle breakpoint** **Debug**, care are efect de comutator. După ce ați lansat în execuție programul (opțiunea **Run** **Run**), execuția lui se va opri la primul punct de întrerupere. Pentru a relua execuția programului alegeți din nou opțiunea **Run** **Run**. Puteți să administrați punctele de întrerupere prin intermediul casetei de dialog **Breakpoints** pe care o deschideți cu opțiunea **Breakpoints...** **Debug**. În caseta de dialog este afișată o listă cu punctele de întrerupere, din care puteți să selectați un punct de întrerupere ca să-l modificați (declanșatorul **Edit**), să-l ștergeți (declanșatorul **Delete**) sau ca să vedeți poziția lui în programul sursă (declanșatorul **View**). Cu declanșatorul **At** puteți să creați un punct de întrerupere la o funcție definită în program.

Funcții utile (fișierul antet **conio.h**)

clrscr() Șterge informațiile de pe ecran. Este bine să scrieți această funcție la începutul programului, pentru ca pe ecran să fie afișate numai rezultatele obținute în urma executării programului.

getch() Așteaptă introducerea unui caracter de la tastatură, pe care nu-l va afișa pe ecran. Rezultatul furnizat este de tip **int** și este codul caracterului introdus. Este bine să scrieți această funcție la sfârșitul programului. În aplicația **Bc**, după ce se termină execuția programului, se revine imediat la interfața mediului de programare și nu mai puteți vedea rezultatele. Cu această funcție, după ce vor fi afișate rezultatele, programul va aștepta să apăsați o tastă pentru a-și încheia execuția și veți putea vedea rezultatele.



Nu uitați, atunci când scrieți programul sursă, să includeți fișierele antet corespunzătoare.

Fișiere antet

Fișierul antet	Necesar pentru
iostream.h	fluxurile de date standard cin>> și cout<< ; constanta endl
conio.h	funcțiile clrscr() , getch()
fstream.h	fluxurile de date pentru fișiere fstream , și funcțiile și constantele pentru fluxurile de date ale fișierelor: eof() , close() , tellp() , seekp() , clear() , get() , getline()
stdio.h	macrocomenzi pentru manipularea fișierelor rename() , remove()
ioanip.h	manipulatori: setw() , setprecision() , setfill() , setiosflags() , resetiosflags() , oct() , dec() , hex() și constante pentru manipulatori
math.h	funcții matematice: abs() , floor() , ceil() , sqrt() , pow() , pow10() , sin() , cos() , tan()
stdlib.h	funcții pentru generarea numerelor aleatoare: randomize() , rand()

Sistem

Reprezentarea simbolurilor numerice.

Un sistem următor

- a) Utilizator de
- b) Program
- c) Utilizator
- d) În cre

Sistem

1. Sistem

- ✓ Utilizator
- ✓ Dezvoltator

2. Sistem

- ✓ Utilizator
- ✓ Dezvoltator

3. Sistem

- ✓ Utilizator
- ✓ Dezvoltator

4. Sistem

- ✓ Utilizator
- ✓ Dezvoltator

10	2
0	0
1	1
2	10
3	11

Pentru a reprezenta

Anexa 2

Sisteme de numerație

Reprezentarea numerelor se face cu ajutorul unor semne grafice numite **cifre**. Totalitatea simbolurilor grafice folosite pentru scrierea numerelor formează **alfabetul sistemului de numerație**. Baza sistemului de numerație este egală cu numărul de simboluri ale alfabetului.

Un **sistem de numerație în baza q** este un sistem de reprezentare a numerelor care are următoarele caracteristici:

- Utilizează un alfabet cu **q** simboluri diferite între ele, numite cifre, care formează un șir de numere consecutive.
- Prima cifră din șir este **0**.
- Ultima cifră din șir este cu o unitate mai mică decât baza: **q-1**.
- În funcție de poziția lor în reprezentarea numărului, cifrele se înmulțesc cu puteri creșcătoare ale bazei, obținându-se dezvoltarea numărului după puterile bazei:

$$n(q) = a_n a_{n-1} \dots a_1 a_0 = a_n \times q^n + a_{n-1} \times q^{n-1} + \dots + a_1 \times q^1 + a_0 \times q^0$$

Sistemele de numerație utile în implementarea algoritmilor cu ajutorul calculatorului sunt:

1. Sistemul zecimal (în baza 10):

- ✓ Utilizează un alfabet cu **10** simboluri, diferite între ele: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9.
- ✓ Dezvoltarea numărului după puterile bazei este:

$$14597_{(10)} = 1 \times 10^4 + 4 \times 10^3 + 5 \times 10^2 + 9 \times 10^1 + 7 \times 10^0$$

2. Sistemul binar (în baza 2):

- ✓ Utilizează un alfabet cu **2** simboluri, diferite între ele: 0, 1.
- ✓ Dezvoltarea numărului după puterile bazei este:

$$100101_{(2)} = 1 \times 2^5 + 0 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$$

3. Sistemul octal (în baza 8):

- ✓ Utilizează un alfabet cu **8** simboluri, diferite între ele: 0, 1, 2, 3, 4, 5, 6, 7.
- ✓ Dezvoltarea numărului după puterile bazei este:

$$173451_{(8)} = 1 \times 8^5 + 7 \times 8^4 + 3 \times 8^3 + 4 \times 8^2 + 5 \times 8^1 + 1 \times 8^0$$

4. Sistemul hexazecimal (în baza 16):

- ✓ Utilizează un alfabet cu **16** simboluri, diferite între ele: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F (A=10₍₁₀₎, B=11₍₁₀₎, C=12₍₁₀₎, D=13₍₁₀₎, E=14₍₁₀₎, F=15₍₁₀₎).
- ✓ Dezvoltarea numărului după puterile bazei este:

$$1AE2_{(16)} = 1 \times 16^3 + A \times 16^2 + E \times 16^1 + 2 \times 16^0$$

10	2	8	16
0	0	0	0
1	1	1	1
2	10	2	2
3	11	3	3

10	2	8	16
4	100	4	4
5	101	5	5
6	110	6	6
7	111	7	7

10	2	8	16
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B

10	2	8	16
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F

Pentru a compara două numere scrise în două baze de numerație diferite **p** și **q**, ele trebuie reprezentate în aceeași bază: fie în baza **p**, fie în baza **q**. Trecerea de la reprezentarea nu-

mărilor în baza q la reprezentarea numărului în baza p se numește **conversia numărului din baza q în baza p** .

Conversia din baza q în baza 10

Conversia se execută astfel:

- Se scrie dezvoltarea numărului după puterile bazei.
- Se înlocuiesc cifrele numărului cu valoarea lor în sistemul zecimal.
- Se efectuează înmulțirile și adunările, în sistemul zecimal.

Exemple:

$$100101_{(2)} = 1 \times 2^5 + 0 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 32 + 4 + 1 = 37_{(10)}$$

$$154_{(8)} = 1 \times 8^2 + 5 \times 8^1 + 4 \times 8^0 = 64 + 40 + 4 = 108_{(10)}$$

$$1AC_{(16)} = 1 \times 16^2 + A \times 16^1 + C \times 16^0 = 1 \times 256 + 10 \times 16 + 12 = 428_{(10)}$$

Conversia din baza 10 în baza q

Conversia se execută astfel:

- Se împarte numărul reprezentat în baza 10 la q . Se obține câtul c_0 și restul r_0 .
- Se împarte c_0 la q . Se obține câtul c_1 și restul r_1 .
- Procesul de împărțire a câtului obținut (c_{n-1}) la q continuă până când câtul obținut c_n este mai mic decât baza q și se obține restul r_n .
- Resturile obținute se convertește separat, fiecare, în baza q , și se formează cu ele numărul reprezentat în baza q : $r_n r_{n-1} \dots r_1 r_0$.

Exemple:

2	cât	rest
29	14	1
14	7	0
7	3	1
3	1	1
1 < 2		
$29_{(10)} = 11101_{(2)}$		

8	cât	rest
7584	948	0
948	118	4
118	14	6
14	1	6
1 < 8		
$7584_{(10)} = 16640_{(8)}$		

16	cât	rest
3508	219	4
219	13 → D	11 → B
13 < 16		
$3508_{(16)} = DB4_{(16)}$		

Conversia din baza 2 în baza 16 (8) și invers

Pentru a converti un număr binar într-un număr hexazecimal (octal) se împart cifrele numărului binar în grupe de câte patru (trei), de la dreapta la stânga, și fiecare grupă se convertește separat într-o cifră hexazecimală (octală).

Pentru a converti un număr hexazecimal (octal) într-un număr binar se convertește fiecare cifră a numărului într-un grup de patru (trei) cifre binare.

Exemple

$$111011111001_{(2)} = 1 \ 1101 \ 1111 \ 1001_{(2)} = 1DF9_{(16)}$$

$$111011111001_{(2)} = 1 \ 110 \ 111 \ 111 \ 001_{(2)} = 16771_{(8)}$$

$$F2A1_{(16)} = 1111 \ 0010 \ 1010 \ 0001_{(2)} = 1111001010100001_{(2)}$$

$$7342_{(8)} = 111 \ 011 \ 100 \ 010_{(2)} = 111011100010_{(2)}$$

Anexa 3

Reprezentarea internă a datelor

Calculatorul prelucrează **date**. Reprezentarea internă a datelor se face diferențiat, în funcție de tipul datei. **Tipul datei** corespunde unui anumit model de reprezentare internă.

Reprezentarea caracterelor

Individual, fiecare caracter (literă, cifră, spațiu sau caracter special) este codificat într-o secvență de lungime fixă de 8 cifre binare, folosind codul standardizat ASCII. El permite construirea a $2^8 = 256$ cuvinte de cod diferite între ele. Fiecărui caracter de pe tastatură îi este atribuită o secvență de cod ASCII prin care poate fi reprezentat în memoria calculatorului. Astfel, caracterul A va fi reprezentat prin secvența de 8 cifre binare 01000001, iar caracterul 9 prin secvența de 8 cifre binare 00111001.

Reprezentarea numerelor

Reprezentarea internă a datelor numerice se face diferențiat, în funcție de tipul lor: numere întregi (cu semn sau fără semn) și numere reale.

Reprezentarea numerelor întregi fără semn

Fiecare număr întreg este codificat cu un număr binar de lungime fixă. Lungimea secvenței binare trebuie să fie multiplu de 8 biți: 8 biți, 16 biți sau 32 biți. Pentru fiecare număr, indiferent de mărimea numărului, sunt folosite secvențe de biți de aceeași lungime. Pentru a obține același număr de biți, sunt adăugate zerouri ne semnificative.

De exemplu, dacă se reprezintă numărul 9 ($9_{(10)} = 1001_{(2)}$) folosind un spațiu de memorare de 16 biți, reprezentarea numărului va fi:

0000 0000 0000 1001



reprezentarea în binar a numărului 9
biți nesemnificativi, pentru completarea reprezentării

Cu cei 16 biți, numărul cel mai mare care se poate reprezenta este:

1111 1111 1111 1111

16 cifre binare

Acest număr este $2^{16} - 1 = 65\,535$. Dacă reprezentarea se va face folosind doar 8 cifre binare, cel mai mare număr va fi $2^8 - 1 = 255$.

Așadar, cu ajutorul a n cifre binare, folosind reprezentarea în binar se pot reprezenta numere întregi fără semn, N , din domeniul: $0 \leq N \leq 2^n - 1$

Folosind această formulă de calcul se poate calcula domeniul de definiție pentru reprezentarea unui număr întreg cu semn pe 8 biți, pe 16 biți sau pe 32 de biți.

Domeniul de definiție al unei date de tip numeric întreg fără semn, reprezentată cu 8 cifre binare (1 octet), va fi 0 ... 255, pentru cea reprezentată cu 16 cifre binare (2 octeți), va fi 0 ... 65 535, iar pentru cea reprezentată cu 32 cifre binare (4 octeți), va fi 0 ... 4 294 967 205.

Atenție! Numărul întreg 9 nu este codificat în același mod cu caracterul cifră 9.

Reprezentarea numerelor întregi cu semn

Modelul de reprezentare internă a datelor numerice întregi cu semn se numește **reprezentarea numerelor prin complementul față de 2**.

Se notează cu N_n^1 complementul față de 2 al unui număr întreg negativ N_n (reprezentat prin n cifre binare). El este $2^n - N_n$.

De exemplu, complementul față de 2 al numărului -6, într-o reprezentare cu 16 cifre binare, se calculează astfel:

$$2^{16} \rightarrow 1\ 0000\ 0000\ 0000\ 0000_{(2)}, \text{ iar } 6_{(10)} = 110_{(2)}$$

$$\begin{array}{r} \text{și:} \quad 1\ 0000\ 0000\ 0000\ 0000 - \\ \quad \quad 0000\ 0000\ 0000\ 0110 = \\ \hline \quad \quad 1111\ 1111\ 1111\ 1010 \end{array}$$

Bitul de pe prima poziție va reprezenta semnul numărului: dacă este 0, numărul este pozitiv, iar dacă este 1, numărul este negativ.

Complementul față de 2 al unui număr întreg negativ se mai poate calcula astfel:

- Se convertește în binar valoarea pozitivă a numărului: 0000 0000 0000 0110;
- se neagă cifrele binare ale numărului obținut: 1111 1111 1111 1001;
- se adună 1 la numărul obținut:

$$\begin{array}{r} 1111\ 1111\ 1111\ 1001 + \\ \quad \quad \quad \quad \quad \quad 1 = \\ \hline 1111\ 1111\ 1111\ 1010 \end{array}$$

Așadar, cu ajutorul a n cifre binare, folosind reprezentarea prin complementul față de 2, se pot reprezenta numere întregi, N , din domeniul: $-2^{n-1} \leq N \leq 2^{n-1}-1$.

Folosind această formulă de calcul se poate calcula domeniul de definiție pentru reprezentarea unui număr întreg cu semn pe 8 biți, pe 16 biți sau pe 32 de biți.

Domeniul de definiție pentru o dată de tip numeric întreg cu semn, reprezentată în complement față de 2, cu 8 cifre binare (1 octet), va fi -128 ... +127, pentru cea reprezentată cu 16 cifre binare (2 octeți) va fi -65 536 ... +65 535, iar pentru cea reprezentată cu 32 cifre binare (4 octeți), va fi -2 147 483 648 ... +2 147 483 647.

Reprezentarea numerelor reale

În această reprezentare, numerele sunt reprezentate prin **exponent** și **mantisă**. Ea se mai numește și **notația științifică**. Se știe că orice număr poate fi scris cu ajutorul puterilor lui 10 (exponenți). În acest mod poate fi controlată poziția virgulei zecimale, iar reprezentarea obținută se numește în virgulă mobilă, deoarece virgula zecimală își poate schimba poziția în funcție de valoarea exponentului. De exemplu, reprezentarea în notație științifică este:

$$12.5 = 12.5 \times 10^0 = 0.125 \times 10^2 = 125 \times 10^{-1} = 125 \text{ E } -1$$

mantisă

exponent

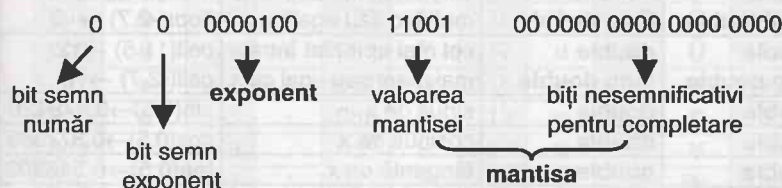
În mod analog, se va reprezenta și intern numărul, codificându-se în binar exponentul și mantisa. În plus, se vor folosi doi biți pentru reprezentarea semnului mantisei și a exponentului. Conform acestei convenții, dacă se consideră reprezentarea în virgulă mobilă pe 32 de biți, biții vor fi folosiți astfel: 1 bit pentru semnul numărului, 1 bit pentru semnul exponentului, 7 biți pentru exponent și 23 biți pentru mantisă.

De exemplu:

$$12,5_{(10)} = 1100,1_{(2)} = 0,11001_{(2)} \times 2^4 = 0,11001_{(2)} \times 10_{(2)}^{100_{(2)}}:$$

- ✓ mantisa este 11001,
- ✓ exponentul este $4_{(10)} = 100_{(2)}$,
- ✓ semnul numărului este pozitiv – 0,
- ✓ semnul mantisei este pozitiv – 0,

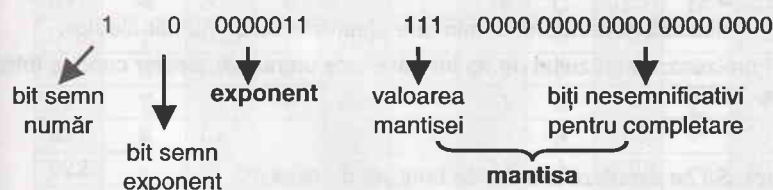
iar reprezentarea numărului este:



$$-7_{(10)} = -111_{(2)} = -0,111_{(2)} \times 2^3 = -0,111_{(2)} \times 10_{(2)}^{11_{(2)}}:$$

- ✓ mantisa este 111,
- ✓ exponentul este $3_{(10)} = 11_{(2)}$,
- ✓ semnul numărului este negativ – 1,
- ✓ semnul mantisei este pozitiv – 0,

iar reprezentarea numărului este:



În această reprezentare s-au folosit 32 de biți, din care 7 pentru reprezentarea exponentului, 23 pentru reprezentarea mantisei. Cel mai mare număr pozitiv care se poate scrie în acest caz se stabilește astfel:

- ✓ Cu 23 cifre binare, cel mai mare număr care se poate scrie este :
 $2^{23} = (2^{10})^{23/10} \approx (1000)^{23/10} = (10^3)^{23/10} = 10^{3 \times 23/10} = 10^{69/10} \approx 10^6$
și numărul maxim de cifre semnificative este 6.

- ✓ Cu 7 cifre binare, cel mare exponent care se poate scrie este $2^7 - 1 = 127$, și factorul de multiplicare cel mai mare va fi:

$$2^{127} = (2^{10})^{127/10} \approx (1000)^{127/10} = (10^3)^{127/10} = 10^{3 \times 127/10} = 10^{381/10} \approx 10^{38}.$$

Așadar, domeniul de valori al datei reprezentate cu 7 cifre binare pentru exponent și 23 de cifre binare pentru mantisă va fi $-10^{38} \dots 10^{38}$, iar data va avea maxim 6 cifre semnificative.

În funcție de numărul de biți folosiți, pentru reprezentarea numărului există: reprezentare în simplă precizie – pe 32 de biți –, și reprezentare în dublă precizie – pe 64 de biți.

Anexa 4

Funcții de sistem utile

Funcții matematice -- fișier antet math.h

Funcția	Tip rezultat	Tip parametri	Furnizează	Exemplu
abs(x)	int	int	modulul lui x	abs(-12) → 12
labs(x)	long	long		abs(12) → 12
fabs(x)	double	double		fabs(-1.2) → 1.2
floor(x)	double	double	cel mai apropiat întreg mai mic sau egal cu x	floor(11.5) → 11
floorl(x)	long double	long double		floor(-2.7) → -3
ceil(x)	double	double	cel mai apropiat întreg mai mare sau egal cu x	ceil(11.5) → 12
ceill(x)	long double	long double		ceil(-2.7) → -2
sin(x)	double	double	sinus de x	sin(0.5) → 0.479426
cos(x)	double	double	cosinus de x	cos(0.5) → 0.877583
tan(x)	double	double	tangentă de x	tan(0.5) → 0.546302
sqrt(x)	double	double	radical de ordinul 2 din x	sqrt(9) → 3
sqrtl(x)	long double	long double		
pow(x,y)	double	double	x la puterea y	pow(2,4) → 16
powl(x,y)	long double	long double		
pow10(x)	double	int	10 la puterea x	pow10(2) → 100
powl10l(x)	long double	int		

Funcții pentru generarea numerelor aleatoare -- fișier antet stdlib.h

randomize() Inițializează generatorul de numere aleatoare cu un număr aleator.

rand() Furnizează un rezultat de tip **int** care este un număr aleator cuprins între 0 și 32767.

Exemplu:

Enunțul problemei: Să se simuleze aruncarea unui zar de cinci ori.

```
#include<iostream.h>
#include <stdlib.h>
void main()
{int i;
randomize();
for (i=1;i<=5;i++) cout<<rand()%6+1<<" ";}
```

Rezolvați:

1. Scrieți câte un program prin care să calculați fiecare dintre funcțiile matematice. Testați programul comparând rezultatul furnizat de el cu rezultatul furnizat de funcția de sistem.
2. Scrieți un program care simulează următorul joc: două persoane aruncă fiecare de trei ori cu zarul. Câștigă persoana care a obținut cele mai multe puncte.
3. Scrieți un program care simulează următorul joc: două persoane extrag la întâmplare, dintr-un pachet de cărți de joc, câte patru cărți fiecare. Câștigă persoana care a extras valori în progresie aritmetică.

Anexa 5

Codul ASCII

Cod		Cod		Cod		Cod		Cod		Cod		Cod	
000	nul	037	%	074	J	111	o	148	ö	185	¶	222	¡
001	☺	038	&	075	K	112	p	149	ò	186	¶	223	■
002	☼	039	'	076	L	113	q	150	û	187	¶	224	α
003	♥	040	(	077	M	114	r	151	ù	188	¶	225	β
004	♦	041	)	078	N	115	s	152	ÿ	189	¶	226	Γ
005	♣	042	*	079	O	116	t	153	Ö	190	¶	227	π
006	♠	043	+	080	P	117	u	154	Ü	191	¶	228	Σ
007	•	044	,	081	Q	118	v	155	ç	192	¶	229	σ
008	■	045	-	082	R	119	w	156	£	193	¶	230	μ
009	○	046	.	083	S	120	x	157	¥	194	¶	231	τ
010	☐	047	/	084	T	121	y	158	Pts	195	¶	232	Φ
011	♂	048	0	085	U	122	z	159	f	196	¶	233	Θ
012	♀	049	1	086	V	123	{	160	á	197	¶	234	Ω
013	♪	050	2	087	W	124		161	ä	198	¶	235	δ
014	♫	051	3	088	X	125	}	162	ó	199	¶	236	∞
015	☼	052	4	089	Y	126	~	163	ú	200	¶	237	Ø
016	▶	053	5	090	Z	127	◊	164	ñ	201	¶	238	ε
017	◀	054	6	091	[	128	Ç	165	Ñ	202	¶	239	∩
018	↕	055	7	092	\	129	ü	166	ª	203	¶	240	≡
019		056	8	093	]	130	é	167	º	204	¶	241	±
020	¶	057	9	094	^	131	â	168	¿	205	¶	242	≥
021	§	058	:	095	_	132	ä	169	ƒ	206	¶	243	≤
022	-	059	;	096	`	133	à	170	¬	207	¶	244	
023	↕	060	<	097	a	134	ä	171	½	208	¶	245	
024	↑	061	=	098	b	135	ç	172	¼	209	¶	246	÷
025	↓	062	>	099	c	136	ê	173	ı	210	¶	247	≈
026	→	063	?	100	d	137	ë	174	«	211	¶	248	°
027	←	064	@	101	e	138	è	175	»	212	¶	249	•
028	ƒ	065	A	102	f	139	ï	176		213	¶	250	-
029	↔	066	B	103	g	140	î	177		214	¶	251	√
030	▲	067	C	104	h	141	ì	178		215	¶	252	∞
031	▼	068	D	105	i	142	Ä	179		216	¶	253	²
032	spațiu	069	E	106	j	143	Å	180	-	217	¶	254	•
033	!	070	F	107	k	144	É	181	=	218	¶	255	
034	"	071	G	108	l	145	æ	182	-	219	¶		
035	.#	072	H	109	m	146	Æ	183	¶	220	¶		
036	\$	073	I	110	n	147	ô	184	¶	221	¶		

Cuprins

1. Informatica și societatea	3
1.1. Prelucrarea informațiilor.....	3
1.2. Informatica	4
1.3. Etapele rezolvării unei probleme	7
1.4. Algoritmul	8
Evaluare	10
2. Datele	12
2.1. Definiția datelor	12
2.1.1. Clasificarea datelor	13
2.1.2. Tipul datelor	19
2.2. Operatorii	21
2.3. Expresiile	26
Evaluare	30
3. Algoritmii	36
3.1. Reprezentarea algoritmilor	36
3.2. Principiile programării structurate	38
3.2.1. Structura liniară.....	38
3.2.2. Structura alternativă.....	39
3.2.3. Structura repetitivă.....	43
3.3. Algoritmi elementari	50
3.3.1. Algoritmi pentru interschimbare.....	50
3.3.2. Algoritmi pentru determinarea maximului (minimului)	52
3.3.3. Algoritmi pentru prelucrarea cifrelor unui număr	54
3.3.4. Algoritmi pentru calcularea c.m.m.d.c.	60
3.3.5. Algoritmi pentru testarea unui număr prim	62
3.3.6. Algoritmi pentru prelucrarea divizorilor unui număr.....	65
3.3.7. Algoritmi pentru conversii între sisteme de numerație	68
3.3.8. Algoritmi pentru generarea șirurilor recurente	71
3.4. Eficiența algoritmilor.....	73
3.5. Aplicarea algoritmilor	78
3.5.1. Rezolvarea problemelor de matematică.....	78
3.5.2. Rezolvarea problemelor de fizică	82
Evaluare	85
4. Implementarea algoritmilor	89
4.1. Caracteristicile limbajului de programare.....	89
4.2. Structura programului	91
4.3. Instrucțiunile declarative	93
4.3.1. Tipuri de date	93
4.3.2. Constantele.....	95
4.3.3. Declararea variabilelor de memorie.....	96
4.3.4. Declararea constantelor simbolice	98

4.3.5. Declararea tipurilor de dată utilizator	99
4.4. Operațiile de citire și scriere	100
Evaluare	104
4.5. Expresia și instrucțiunea expresie	106
4.5.1. Operatorii aritmetici	107
4.5.2. Operatorul pentru conversie implicită	109
4.5.3. Operatorii pentru incrementare și decrementare	111
4.5.4. Operatorii relaționali	113
4.5.5. Operatorii logici	113
4.5.6. Operatorii logici pe biți	114
4.5.7. Operatorii de atribuire	118
4.5.8. Operatorul condițional	121
4.5.9. Operatorul virgulă	123
4.5.10. Operatorul dimensiune	125
4.5.11. Precedența și asociativitatea operatorilor	125
Evaluare	127
4.6. Instrucțiunile de control	131
4.6.1. Instrucțiunea if...else	131
4.6.2. Instrucțiunea switch...case	134
4.6.3. Instrucțiunea while	137
4.6.4. Instrucțiunea for	139
4.6.5. Instrucțiunea do...while	146
Evaluare	149
5. Implementarea structurilor de date	155
5.1. Structurile de date	155
5.2. Tablourile de memorie	158
5.3. Implementarea tablourilor de memorie în limbajul C++	162
5.3.1. Tabloul cu o singură dimensiune – vectorul	162
5.3.2. Tabloul cu două dimensiuni – matricea	164
5.4. Algoritmi pentru prelucrarea tablourilor de memorie	166
5.4.1. Algoritmi pentru parcurgerea tablourilor de memorie	166
5.4.2. Algoritmi pentru căutarea unui element într-un tablou de memorie	174
5.4.3. Algoritm pentru ștergerea unui element dintr-un vector	176
5.4.4. Algoritm pentru inserarea unui element într-un vector	177
5.4.5. Algoritmi pentru sortarea unui vector	179
5.4.6. Algoritm pentru interclasarea a doi vectori	188
5.4.6. Aplicarea algoritmilor pentru prelucrarea tablourilor de memorie	190
Evaluare	193
5.5. Fișierele	201
5.6. Implementarea fișierelor text în limbajul C++	204
5.6.1. Fluxuri de date pentru fișiere text	205
5.6.2. Citiri și scrieri cu format	210
5.6.3. Aplicații cu prelucrări de fișiere	213

Evaluare	219
Anexa 1 – Mediul de programare C++	222
Anexa 2 – Sisteme de numerație	231
Anexa 3 – Reprezentarea internă a datelor	233
Anexa 4 – Funcții de sistem utile	236
Anexa 5 – Codul ASCII	237