MINISTERUL EDUCAŢIEI, CULTURII ŞI CERCETĂRII

INFORMATICĂ

Manual pentru clasa a 11-a

ANATOL GREMALSCHI • IURIE MOCANU
LUDMILA GREMALSCHI

Ştiinţa, 2020

CZU 004(075.3)
G 80

Manualul şcolar a fost realizat în conformitate cu prevederile Curriculumului la disciplină, aprobat prin Ordinul Ministerului Educaţiei, Culturii şi Cercetării nr. 906 din 17 iulie 2019. Manualul a fost aprobat prin Ordinul Ministerului Educaţiei, Culturii şi Cercetării nr. 1219 din 6 noiembrie 2020, urmare a evaluării calităţii metodico-ştiinţifice.

Editat din sursele financiare ale *Fondului Special pentru Manuale.*

Comisia de evaluare:
Svetlana Brînză, profesor, grad didactic superior, Liceul Teoretic „N.M. Spătaru", metodist DGETS, Chișinău (coordonator);
Gheorghe Chistruga, profesor, grad didactic superior, Liceul Teoretic „M. Eminescu", Drochia;
Natalia Schițco, profesor, grad didactic superior, Liceul Teoretic „Socrate", Chișinău;
Violina Bargan, profesor, grad didactic superior, Liceul Teoretic „Gh. Asachi", Chișinău;
Ecaterina Adam, profesor, grad didactic unu, Liceul de Creativitate și Inventică „Prometeu-Prim", Chișinău

Denumirea instituției de învățământ _____

Manualul a fost folosit: _____

Anul de folosire	Numele, prenumele elevului	Anul de studii	Aspectul manualului	
			la primire	la returnare

Dirigintele verifică dacă numele, prenumele elevului sunt scrise corect.

Elevii nu vor face niciun fel de însemnări în manual.

Aspectul manualului (la primire şi la returnare) se va aprecia cu unul dintre următorii termeni: *nou, bun, satisfăcător, nesatisfăcător.*

Responsabil de ediţie: Larisa Dohotaru
Redactor: Mariana Belenciuc
Corector: Maria Cornesco
Redactor tehnic: Nina Duduciuc
Machetare computerizată: Vitalie Ichim
Copertă: Romeo Şveţ

ÎNTREPRINDEREA
EDITORIAL-POLIGRAFICĂ
ȘTIINȚA

str. Academiei, nr. 3; MD-2028, Chișinău, Republica Moldova
tel.: (+373 22) 73-96-16; fax: (+373 22) 73-96-27
e-mail: prini_stiinta@yahoo.com
www.editurastiinta.md

Descrierea CIP a Camerei Naţionale a Cărţii
Gremalschi, Anatol.
Informatică: Manual pentru clasa a 11-a / Anatol Gremalschi, Iurie Mocanu, Ludmila Gremalschi;
Comisia de evaluare: Svetlana Brînză (coordonator) [et al.]; Ministerul Educaţiei, Culturii şi Cercetării. –
Ch.: Î.E.P. *Ştiinţa*, 2020 (Combinatul Poligrafic). – 224 p.: fig., tab.
Proprietate a Min. Educaţiei, Culturii şi Cercet.

ISBN 978-9975-85-246-3

Imprimare la **COMBINATUL POLIGRAFIC**
str. Petru Movilă, 35, MD-2004, Chișinău, Republica Moldova, Comanda nr.

CUPRINS

INTRODUCERE

Dragi prieteni,

Impresionantele realizări în domeniul informaticii, crearea supercalculatoarelor şi a calculatoarelor personale, apariţia cyberspaţiului, a *Internetului* şi a realităţii augmentate presupun o cunoaştere profundă a principiilor de funcţionare şi a structurii calculatoarelor moderne, a metodelor de programare a acestora. Indiferent de specificul viitoarei activităţi profesionale a fiecărui absolvent de liceu, aceste cunoştinţe vor servi drept o călăuză demnă de toată încrederea într-o lume digitală aflată mereu în schimbare.

Dezvoltarea continuă a informaticii ca ştiinţă fundamentală şi dezvoltarea fulminantă a tehnologiei informaţiei şi comunicaţiilor ca ştiinţă aplicativă are drept efect apariţia de noi şi noi echipamente digitale care, practic, se reînnoiesc la fiecare doi-trei ani. „Invazia" gadgeturilor, diversitatea crescândă a produselor-program, apariţia *Internetului Obiectelor* cer revizuirea metodelor de învăţare a informaticii, reorientându-le de la formarea simplelor dexterităţi de acţionare a butoanelor, de clicare şi glisare a obiectele de pe ecranele tactile, la formarea competenţelor de a utiliza şi crea echipamente şi produse digitale noi, cum ar fi programe de aplicaţii, documente Web, servicii digitale online, opere de artă digitală.

În lumea modernă, cele mai impresionante inovaţii apar la intersecţia diverselor domenii ale cunoaşterii. Drept exemplu pot servi digitalizarea radioului şi a televiziunii, a muzicii şi a artelor vizuale, extinderea comerţului online, creşterea rolului reţelelor sociale, afirmarea instruirii asistate de calculator. Apariţia şi dezvoltarea inteligenţei artificiale, care, conform previziunilor mai multor personalităţi ilustre ale zilelor de astăzi, va schimba în esenţă evoluţia civilizaţiei umane, presupune conjugarea eforturilor specialiştilor din diverse domenii de activitate, indiferent de natura acestora.

Prin urmare, indiferent de profilul de liceu pe care îl urmează elevii, umanist sau real, pentru a-şi atinge scopurile de dezvoltare personală, inclusiv pentru o carieră profesională de succes şi o viaţă împlinită, ei neapărat vor avea nevoie de acele competenţe digitale, formarea şi dezvoltarea cărora reprezintă scopul acestui manual. Mai exact, la finele clasei a 11-a elevii vor putea:

— să implementeze într-un limbaj de programare de nivel înalt algoritmii de prelucrare a datelor structurate;

— să utilizeze bazele teoriei informaţiei, aritmeticii de calculator şi a algebrei booleene pentru elaborarea modelelor informatice;

— să interpreteze rezultatele furnizate de programele de calculator elaborate;

— să identifice structura generală a sistemelor digitale utilizate, a principiilor de funcţionare a sistemelor de transmitere, stocare şi de prelucrare a informaţiei.

Studierea unuia dintre modulele la alegere va contribui la valorificarea metodelor şi instrumentelor specifice prelucrărilor digitale, la dezvoltarea gândirii şi a creativităţii în integrarea achiziţiilor informatice cu cele din alte domenii, la conştientizarea importanţei respectării regulilor de securitate, ergonomice şi etice în crearea şi difuzarea produselor digitale.

În scopul autoevaluării, recomandăm elevilor să descarce de pe pagina web http://www.ctice.gov.md a Centrului Tehnologii Informaţionale şi Comunicaţionale în Educaţie testele respective, să rezolve probele din componenţa acestora şi să compare răspunsurile lor cu răspunsurile-etalon, care pot fi obţinute de la profesorii de informatică.

Vă dorim succese,
Autorii

Capitolul 1

TIPURI DE DATE STRUCTURATE

1.1. Date simple şi date structurate

Cunoaştem deja că informația ce urmează să fie supusă unei prelucrări este accesibilă calculatorului în formă de **date**. Datele sunt constituite din cifre, litere, semne, numere, şiruri de caractere ş.a.m.d.

Într-un limbaj cod-calculator datele sunt reprezentate prin secvențe de cifre binare. De exemplu, la nivelul procesorului numărul natural 1 039 se reprezintă în sistemul de numerație binar ca:

```
10000001111
```

Pentru a scuti utilizatorul de toate detaliile legate de reprezentarea internă a datelor, limbajele de programare utilizează diverse tipuri de date. Amintim că prin **tip de date** se înțelege o **mulțime de valori** şi o **mulțime de operații** care pot fi efectuate cu valorile respective.

În clasele precedente ați studiat următoarele tipuri de date:

- `integer`/**`int`**, destinat prelucrării pe calculator a numerelor întregi;
- `real`/**`float`**, pentru prelucrarea numerelor reale;
- `boolean`/**`bool`**, utilizat în cazul prelucrării valorilor de adevăr;
- `char`/**`char`**, destinat reprezentării şi prelucrării caracterelor;
- *enumerare* (**`enum`**), care include o mulțime ordonată de valori specificate prin identificatori;
- *subdomeniu* (doar în limbajul PASCAL), care include o submulțime de valori ale unui tip `integer`, `boolean`, `char` sau *enumerare*.

Din perspectiva limbajelor de programare, numerele întregi, numerele reale, valorile de adevăr, caracterele, valorile ordonate specificate prin identificatori se numesc **date simple**, iar tipurile enumerate mai sus – **tipuri de date simple**.

În general, datele simple nu sunt suficiente pentru a reprezenta pe calculator şi a prelucra în mod eficient informațiile din lumea înconjurătoare.

De exemplu, în cazul prelucrării textelor, operațiile de ştergere, copiere şi mutare se fac nu doar la nivel de caractere (tipul de date simple `char`), ci şi la nivel de cuvinte, propoziții, rânduri, paragrafe şi chiar pagini întregi.

Evident, un cuvânt sau o propoziție pot fi reprezentate ca o succesiune de caractere, adică printr-o structură care este formată dintr-o înşiruire de date simple de tipul `char`. Pentru a descrie astfel de structuri, în limbajele de programare de nivel înalt se folosesc tipuri de date denumite sugestiv *şiruri de caractere*. În consecință, literele alfabetului latin `A`, `a`, `B`, `b`, `C`, `c`, …, `Z`, `z` se reprezintă pe

calculator prin datele simple de tipul `char`, pe când cuvintele formate din ele, de exemplu, `Liceu`, `Patrie`, `Informatica` etc. – prin date structurate de tipul *şir de caractere*.

Un alt exemplu ce ilustrează necesitatea utilizării datelor structurate constă în prelucrarea informațiilor referitoare la consumul zilnic de energie electrică al unei gospodării. Acest consum, în kWh, poate fi reprezentat pe calculator printr-un număr real (tipul de date simple `real` sau **`float`**). Dacă ne dorim însă să analizăm variațiile consumului zilnic de energie electrică pe durata unei luni, va trebui să reprezentăm datele respective printr-un tabel unidimensional, format dintr-un singur rând:

Ziua lunii	*1*	*2*	*3*	*...*	*31*
Consumul zilnic, kWh	13,4	18,6	9,4	…	15,0

În prima celulă a acestui tabel se va conține consumul în kWh din prima zi a lunii; în celula a doua – consumul din ziua a doua a lunii ş.a.m.d., până la celula 31, în funcție de numărul de zile ale lunii în cauză.

În cazul analizei consumului zilnic de energie electrică pe parcursul unui an, va trebui să folosim un tabel format din 12 rânduri şi 31 de coloane, adică un tabel bidimensional:

Ziua lunii	*1*	*2*	*3*	*...*	*31*
Ianuarie	13,4	18,6	9,4	…	15,0
Februarie	12,9	14,3	21,7	…	…
…	…	…	…	…	…
Decembrie	11,7	10,4	3,4	…	25,3

Primul rând al unui astfel de tabel va conține consumul pentru fiecare din zilele lunii ianuarie; rândul al doilea – consumul pentru fiecare din zilele lunii februarie ş.a.m.d., până la rândul ce corespunde lunii decembrie.

Se observă că tabelele de mai sus reprezintă date structurate, fiind formate din date simple de tipul `real`/**`float`**. De obicei, pe calculator tabelele respective se reprezintă cu ajutorul unor tipuri speciale de date structurate, denumite *tablouri*.

Accentuăm faptul că tablourile conțin doar datele din celule, fără denumirile de rânduri şi de coloane. Pe calculator, aceste denumiri sunt reprezentate prin alte mărimi, denumite *indici*. Evident, în cazul tabelelor unidimensionale este necesar doar un singur indice, iar în cazul tabelelor bidimensionale – de doi.

De exemplu, în cazul tabloului unidimensional ce reprezintă pe calculator consumul zilnic de energie electrică pe parcursul unei luni, în calitate de indice se foloseşte o mărime de tip întreg, care poate lua doar valorile *1*, *2*, *3*, *…*, *31*.

În cazul tabloului bidimensional ce reprezintă consumul zilnic de energie electrică pe parcursul unui an, sunt necesari doi indici, unul pentru rânduri şi altul pentru coloane. În calitate de indice pentru rânduri se foloseşte o mărime de tip enumerare, care ia valorile *Ianuarie*, *Februarie*, *Martie*, *…*, *Decembrie*. Pentru indicele de coloane se foloseşte o mărime de tip întreg care ia valorile *1*, *2*, *3*, *… 31*.

Datele formate prin agregarea (unirea într-un tot) a unor date simple se numesc *date structurate*. Tipurile de date utilizate pentru definirea unor astfel de date se numesc *tipuri de date structurate*.

În afară de tipurile de date structurate *şiruri de caractere* şi *tablouri*, descrise în linii generale mai sus, limbajele de programare oferă informaticienilor posibilitatea să utilizeze şi alte tipuri de date structurate, cele mai des utilizate fiind *articolele*, *mulțimile* şi *fişierele*.

Întrebări şi exerciţii

❶ Explicați semnificația termenului *tip de date*. Dați exemple.

❷ Care este diferența dintre datele simple şi datele structurate?

❸ Dați exemple de tipuri de date simple şi tipuri de date structurate.

❹ OBSERVĂ! Determinați tipul datelor, simple sau structurate, din exemplele ce urmează:

a)

138

b)

3.14

c)

Munteanu Elena

d)

Clasa a 11-a

e)

12	6	231	5

f)

32.51	149,28	318,56	20013.9	0.4536	721.3

g)

4	635	-8	+27
72	41	319	432
16	-20	45	1830

În cazul datelor structurate, indicați tipul datelor simple din componența acestora.

Modele de răspuns:

– Dată simplă, de tip *număr întreg*.

– Dată structurată de tip *tablou unidimensional*, format din numere întregi.

❺ CREEAZĂ! Elaborați un tabel unidimensional ce conține plățile lunare pentru energia electrică folosită de familia voastră pe parcursul ultimului an calendaristic. Folosiți în acest scop informațiile din facturile de plată emise de compania ce vă furnizează energia electrică. Ce tipuri de date veți utiliza pentru reprezentarea pe calculator a informațiilor din acest tabel?

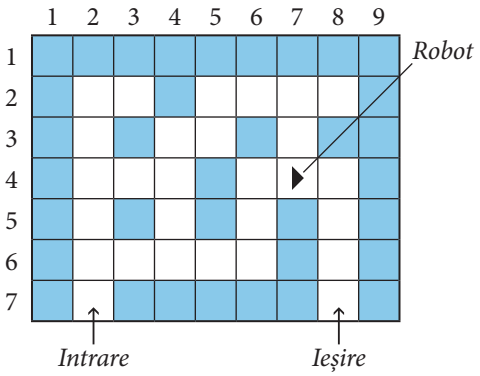
❻ EXPLOREAZĂ! Site-ul Băncii Naționale a Moldovei conține informații referitoare la ratele oficiale medii de schimb (cursul valutar). De exemplu, în luna ianuarie a anului 2019, rata medie oficială de schimb pentru moneda unică a Uniunii Europene era: 1 Euro = 19,6501 Lei Moldoveneşti. Elaborați:

– un tabel unidimensional ce conține ratele lunare medii oficiale de schimb pentru ultimul an calendaristic;

– un tabel bidimensional ce conține ratele lunare medii oficiale de schimb pentru ultimii trei ani.

Ce tipuri de date veți utiliza pentru reprezentarea pe calculator a informațiilor din aceste tabele?

❼ CREEAZĂ! După o vizită la renumitele hrube de la Cricova, vizită pe durata căreia a avut posibilitatea să călătorească prin galeriile subterane ale acestui important obiect turistic, un informatician a construit un robot care găseşte cel mai scurt drum de la o sală de expoziție la alta. În memoria calculatorului ce dirijează robotul, hrubele sunt reprezentate printr-un dreptunghi divizat în pătrățele (vezi figura de mai jos).



Pătrățelele haşurate reprezintă obstacolele (pereții galeriilor şi sălilor de expoziție, standurile cu exponate), iar cele nehaşurate – spațiile libere. Robotul poate executa doar instrucțiunile SUS, JOS, DREAPTA, STANGA, conform cărora se deplasează în unul din pătrățelele vecine. Dacă în acest pătrat este un obstacol, de exemplu, un perete sau un stand, robotul se opreşte şi nu mai reacționează la nicio instrucțiune.

Elaborați structura de date necesară pentru reprezentarea planului hrubelor de la Cricova pe calculator. Asigurați-vă că structura elaborată conține toate informațiile de pe plan: obstacolele, spațiile libere, pătrățelul în care se află robotul, intrarea, ieşirea.

❽ ÎNVAȚĂ SĂ ÎNVEȚI! Cunoaşteți deja că datele structurate se obțin prin agregarea (unirea într-un tot) a unor date simple. Limbajele PASCAL şi C++ permit agregarea nu doar a datelor simple, dar şi a celor structurate. De exemplu, în celulele unui tabel pot fi memorate nu doar date simple (numere întregi, numere reale, caractere etc.), dar şi date structurate, cum ar fi şirurile de caractere.

Se consideră lista elevilor unei clase de liceu:

```
Munteanu Ion
Florea Elena
…
Prisăcaru Alexandra
```

Elaborați structura de date necesară pentru reprezentarea unei astfel de liste pe calculator. Asigurați-vă că structura elaborată permite sortarea elevilor în ordine alfabetică.

1.2. Tipuri de date *tablou*

PASCAL

Pentru definirea tipurilor de date tablou, în limbajul PASCAL se utilizează cuvântul-cheie **array**. Tablourile sunt formate dintr-un număr fixat de componente de acelaşi tip, denumit tip **de bază**. Referirea componentelor se face cu ajutorul unui sau a mai multor **indici**.

Un tip de date *tablou unidimensional* se defineşte printr-o construcţie de forma

type <*Nume tip*> = **array** [T_1] **of** T_2;

unde T_1 este tipul indicelui care trebuie să fie ordinal, iar T_2 este tipul componentelor (tipul de bază) care poate fi un tip oarecare.

Exemple:

1)
```
type Vector = array [1..5] of real;
var  x : Vector;
```

2)
```
type Zi = (L, Ma, Mi, J, V, S, D);
     Venit = array [Zi] of real;
var  v : Venit;
     z : Zi;
```

3)
```
type Ora = 0..23;
     Grade = -40..40;
     Temperatura = array [Ora] of Grade;
var  t : Temperatura;
     h : Ora;
```

Structura datelor din exemplele în studiu este prezentată în *figura 1.1.*

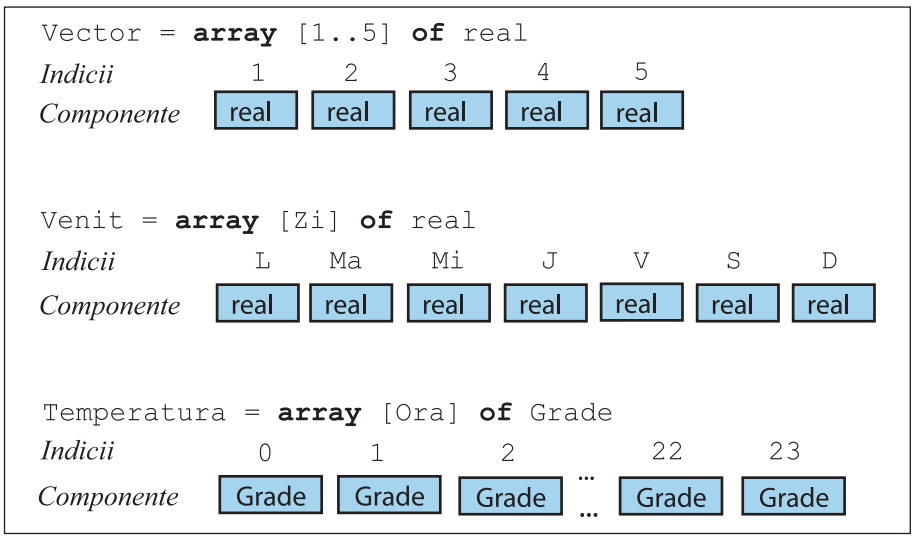


Fig. 1.1. Structura datelor de tip `Vector`, `Venit` şi `Temperatura`

Fiecare componentă a unei variabile de tip *tablou unidimensional* poate fi specificată explicit, prin numele variabilei urmat de indicele respectiv încadrat de paranteze pătrate.

Exemple:

1) `x[1], x[4];`

2) `v[L], v[Ma], v[J];`

3) `t[0], t[15], t[23];`

4) `v[z], t[h].`

Asupra componentelor datelor de tip *tablou* se pot efectua toate operațiile admise de tipul de bază respectiv. Programul ce urmează afișează pe ecran suma componentelor variabilei `x` de tip `Vector`. Valorile componentelor `x[1], x[2], ..., x[5]` se citesc de la tastatură.

```
Program P78;
{ Suma componentelor variabilei x de tip Vector }
{ Numar fix de componente
type Vector = array [1..5] of real;
var x : Vector;
    i : integer;
    s : real;
begin
  writeln('Dati 5 numere:');
  for i:=1 to 5 do readln(x[i]);
  writeln('Ati introdus:');
  for i:=1 to 5 do writeln(x[i]);
  s:=0;
  for i:=1 to 5 do s:=s+x[i];
  writeln('Suma=', s);
  readln;
end.
```

Pentru a extinde aria de aplicare a unui program, se recomandă ca numărul de componente ale datelor de tip **`array`** să fie specificate prin constante.

De exemplu, programul `P78` poate fi modificat pentru a însuma n numere reale, $n \leq 100$:

```
Program P79;
{ Suma componentelor variabilei x de tip Vector }
{ Numar flotant de componente
const nmax = 100;
type Vector = array [1..nmax] of real;
var x : Vector;
    n : 1..nmax;
```

```
    i : integer;
    s : real;
begin
  write('n='); readln(n);
  writeln('Dati ', n, ' numere:');
  for i:=1 to n do readln(x[i]);
  writeln('Ati introdus:');
  for i:=1 to n do writeln(x[i]);
  s:=0;
  for i:=1 to n do
    s:=s+x[i];
  writeln('Suma=', s);
  readln;
end.
```

În informatică, tablourile unidimensionale sunt utilizate foarte des pentru a sorta datele într-o anumită ordine, de exemplu, în cea crescătoare. Pentru o astfel de sortare se foloseşte următoarea metodă:

1) fiecare componentă a tabloului, începând cu prima, este comparată consecutiv cu fiecare dintre componentele ce urmează după ea;

2) dacă în urma comparării se constată că componenta curentă este mai mare decât cea cu care este comparată, ele sunt schimbate cu locul;

3) procesul de parcurgere a tabloului continuă până când vor fi comparate ultima şi penultima dintre componente.

În limbajul cotidian al informaticienilor, o astfel de sortare se numeşte *sortare prin metoda bulelor*, întrucât componentele tabloului îşi schimbă poziţiile ca bulele dintr-un lichid: cele mai uşoare se ridică la suprafaţă, iar cele mai grele se duc la fund.

Pentru exemplificare, în programul ce urmează metoda bulelor este folosită pentru sortarea în ordine crescătoare a componentelor tabloului A. Componentele acestui tablou sunt citite de la tastatură, iar sortarea propriu-zisă se face în tabloul B.

```
Program P80;
 { Sortarea prin metoda bulelor }
const nmax=100;
type Tablou = array[1..nmax] of integer;
var A, B : Tablou;
    n, i, j : integer;
    x : integer;
begin
  write(,Dati numarul de componente n= ,);
  readln(n);
  writeln(,Dati componentele tabloului A:');
  for i:=1 to n do read(A[i]);
  readln;
```

```
    B:=A;
    for i:=1 to n-1 do
      for j:=i+1 to n do
        if (B[i]>B[j]) then
          begin
            x:=B[i];
            B[i]:=B[j];
            B[j]:=x;
          end;
    writeln(,Tabloul initial A:');
    for i:=1 to n do write(A[i], , ,);
    writeln;
    writeln(,Tabloul sortat B:');
    for i:=1 to n do write(B[i], , ,);
    writeln;
    readln;
end.
```

Un tip de date *tablou bidimensional* se defineşte cu ajutorul construcției:

type <*Nume tip*> **= array** [T_1, T_2] **of** T_3;

unde T_1 şi T_2 specifică tipul indicilor, iar T_3 – tipul componentelor.

Pentru exemplificare, în *figura 1.2* este prezentată structura datelor tipului `Matrice`:

```
type Matrice = array [1..3, 1..4] of real;
```

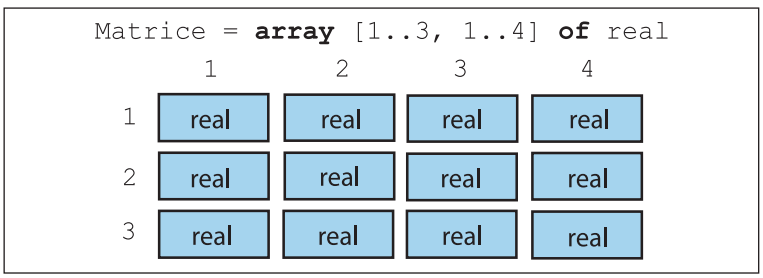


Fig. 1.2. Structura datelor de tip `Matrice`

Componentele unei variabile de tip *tablou bidimensional* se specifică explicit prin numele variabilei urmat de indicii respectivi separați prin virgulă şi încadrați de paranteze pătrate.

De exemplu, în prezența declarației

```
var m : Matrice;
```

notația `M[1,1]` specifică componenta din linia 1, coloana 1 (*fig. 1.2*); notația `M[1,2]` specifică componenta din linia 1, coloana 2; notația `M[i,j]` specifică componenta din linia `i`, coloana `j`.

Programul ce urmează afişează pe ecran suma componentelor variabilei `M` de tip `Matrice`. Valorile componentelor `M[1,1]`, `M[1,2]`, ..., `M[3,4]` se citesc de la tastatură.

```
Program P81;
{ Suma componentelor variabilei M de tip Matrice }
type Matrice = array [1..3, 1..4] of real;
var M : Matrice;
    i, j : integer;
    s : real;
begin
  writeln('Dati componentele M[i,j]:');
  for i:=1 to 3 do
    for j:=1 to 4 do
      begin
        write('M[', i, ',', j, ']=');
        readln(M[i,j]);
      end;
  writeln('Ati introdus:');
  for i:=1 to 3 do
    begin
      for j:=1 to 4 do write(M[i,j]);
      writeln;
    end;
  S:=0;
  for i:=1 to 3 do
    for j:=1 to 4 do
      S:=S+M[i,j];
  writeln('Suma=', S);
  readln;
end.
```

În general, un tip **tablou *n*-dimensional** (n = 1, 2, 3 etc.) se defineşte cu ajutorul diagramelor sintactice din *figura 1.3*. Atributul **`packed`** (împachetat) indică cerința de optimizare a spațiului de memorie pentru elementele tipului **`array`**. Menționăm că în majoritatea compilatoarelor actuale utilizarea acestui atribut nu are niciun efect, întrucât optimizarea se efectuează în mod automat.

Fiind date două variabile de tip tablou de acelaşi tip, numele variabilelor pot apărea într-o instrucțiune de atribuire. Această atribuire înseamnă copierea tuturor componentelor din membrul drept în membrul stâng.

De exemplu, în prezența declarațiilor

```
var a, b : Matrice;
```

instrucțiunea

```
a:=b
```

este corectă.

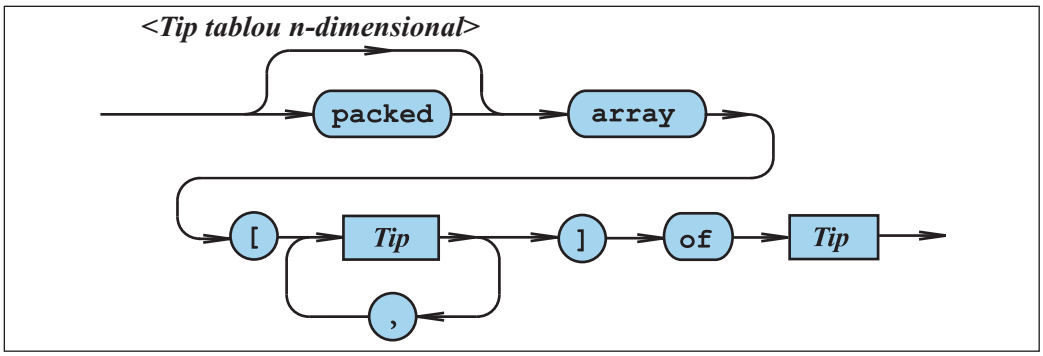


Fig. 1.3. Diagrama sintactică <*Tip tablou n-dimensional*>

În exemplele de mai sus tipul de bază (tipul componentelor) a fost de fiecare dată un tip simplu. Deoarece tipul de bază poate fi, în general, un tip arbitrar, devine posibilă definirea tablourilor cu componente de tip structurat. Considerăm acum un exemplu în care tipul de bază este el însuşi un tip **array**:

```
Type Linie = array [1..4] of real;
     Tabel = array [1..3] of Linie;
var L : Linie;
    T : Tabel;
    x : real;
```

Variabila `T` este formată din 3 componente: `T[1]`, `T[2]` şi `T[3]` de tipul `Linie`. Variabila `L` este formată din 4 componente: `L[1]`, `L[2]`, `L[3]` şi `L[4]` de tipul `real`.

Prin urmare, atribuirile

```
L[1]:=x; x:=L[3]; T[2]:=L; L:=T[1]
```

sunt corecte.

Elementele variabilei `T` pot fi specificate prin `T[i][j]` sau prescurtat `T[i,j]`. Aici `i` indică numărul componentei de tip `Linie` în cadrul variabilei `T`, iar `j` – numărul componentei de tip `real` în cadrul componentei `T[i]` de tip `Linie`.

Subliniem faptul că declarațiile de forma

type array [T_1, T_2] **of** T_3

şi

type array [T_1] **of array** [T_2] **of** T_3

definesc tipuri distincte de date. Prima declarație definește tablouri bidimensionale cu componente de tipul T_3. A doua declarație definește tablouri unidimensionale cu componente de tipul **array** [T_2] **of** T_3.

În programele PASCAL tablourile se utilizează pentru a grupa sub un singur nume mai multe variabile cu caracteristici identice.

C++

În limbajul de programare C++, un tip de date *tablou unidimensional* se definește printr-o construcție de forma:

typedef <*Tip componente*> <*Nume tip tablou*> [<*Număr componente*>];

În C++, un tip de date *tablou unidimensional* se defineşte printr-o construcție de forma

<*Tip*> <*Nume_tablou*> [<Nr_componente>];

unde <*Tip componente*> este tipul componentelor tabloului, care poate fi aproape orice tip de date, iar <*Număr componente*> indică numărul de componente ale tabloului.

Indicii componentelor tabloului pot lua doar valori consecutive întregi, începând cu zero: 0, 1, 2, …, <*Număr componente*> –1.

Exemple:

1)
```
itypedef int Vector[5];
Vector V;
```

2)
```
typedef char Simbol[10];
Simbol S;
```

3)
```
typedef float Acceleratie[45];
Acceleratie A;
```

Structura datelor din exemplele în studiu este prezentată în *figura 1.1**.

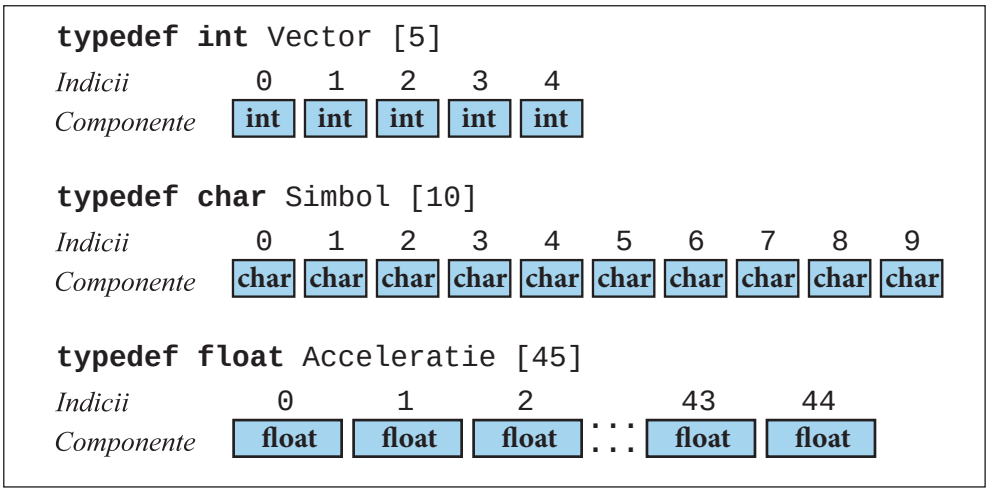


*Fig. 1.1** Structura datelor de tip Vector, Simbol și Acceleratie

Fiecare componentă a unei variabile de tip *tablou unidimensional* poate fi specificată explicit, prin numele variabilei urmat de indicele respectiv încadrat de paranteze pătrate. Indicii trebuie să fie expresii întregi.

Exemple:

```
1)V[1], V[4];
```

```
2)S[0], S[5], S[9];
```

```
3)A[0], A[44], A[23];
```

Asupra componentelor datelor de tip *tablou unidimensional* se pot efectua toate operațiile admise de tipul acestora. Programul ce urmează afişează pe ecran suma componentelor variabilei V de tip tablou unidimensional. Valorile componentelor V[0], V[1], ..., V[4] se citesc de la tastatură.

```
// Program P78
// Suma componentelor variabilei V de tip Vector
// Numar fix de componente
#include <iostream>
using namespace std;
int main()
{
  typedef int Vector[5];
  Vector V;
  int i, S;
  cout <<"Dati 5 numere intregi:" << endl;
  for (i=0; i<5; i++) cin>>V[i];
  cout << "Ati introdus:" << endl;
  for (i=0; i<5; i++) cout << V[i] << ' ';
  cout << endl;
  S=0;
  for (i=0; i<5; i++) S=S+V[i];
  cout << "Suma= " << S;
  return 0;
}
```

Pentru a extinde aria de aplicare a unui program, se recomandă ca numărul de componente ale datelor de tip *tablou* să fie specificate prin constante. Acest fapt oferă o flexibilitate mai mare programelor C++, ele putând fi aplicate pentru prelucrarea tablourilor, numărul de componente al cărora este necunoscut la momentul scrierii programului.

De exemplu, programul P78 poate fi modificat pentru a însuma n numere întregi, $n \leq 100$. Numărul concret de valori n ce vor fi stocate în tabloul unidimensional V se citeşte de la tastatură.

```
//Program P79
// Suma componentelor variabilei V de tip Vector
// Numar flotant de componente
#include <iostream>
using namespace std;
int main()
{
  const int nmax=100;
  typedef int Vector[nmax];
  Vector V;
  int n, i, S;
```

```
    cout << "Dati n= "; cin >> n;
    cout << "Dati "<<n<<" numere intregi:" << endl;
    for (i=0; i<n; i++) cin >> V[i];
    cout << "Ati introdus:" << endl;
    for (i=0; i<n; i++) cout << V[i] << ' ';
    cout << endl;
    S=0;
    for (i=0; i<n; i++) S=S+V[i];
    cout << "Suma= " << S;
    return 0;
}
```

În informatică, tablourile unidimensionale sunt utilizate foarte des pentru a sorta datele într-o anumită ordine, de exemplu, în cea crescătoare. Pentru o astfel de sortare se foloseşte următoarea metodă:

1) fiecare componentă a tabloului, începând cu prima, este comparată consecutiv cu fiecare dintre componentele ce urmează după ea;

2) dacă în urma comparării componentelor se constată că cea curentă este mai mare decât cea cu care este comparată, ele sunt schimbate cu locul;

3) procesul de parcurgere a tabloului continuă până când vor fi comparate ultima şi penultima dintre componente.

În limbajul cotidian al informaticienilor, o astfel de sortare se numeşte *sortare prin metoda bulelor*, întrucât componentele tabloului îşi schimbă poziţiile ca bulele dintr-un lichid: cele mai uşoare se ridică la suprafaţă, iar cele mai grele se duc la fund.

Pentru exemplificare, în programul ce urmează, metoda bulelor este folosită pentru sortarea în ordine crescătoare a componentelor tabloului A. Componentele acestui tablou sunt citite de la tastatură, iar sortarea propriu-zisă se face în tabloul B.

```
// Program P80
// Sortarea prin metoda bulelor
#include <iostream>
using namespace std;
int main()
{
  const int nmax= 100;
  typedef int Tablou[nmax];
  Tablou A, B;
  int n, i, j;
  int x;
  cout<<"Dati numarul de componente n= ";
  cin>>n;
  cout<<"Dati componentele tabloului A: \n";
  for (i=0; i<n; i++)  cin>>A[i];
```

```
    for (i=0; i<n; i++)  B[i]=A[i];
    for (i=0; i<n-1; i++)
      for (j=i+1; j<n; j++)
        if (B[i]>B[j])
          { x=B[i]; B[i]=B[j]; B[j]=x;}
    cout<<"Tabloul initial: \n";
    for (i=0; i<n; i++) cout<<A[i]<<" ";
    cout << endl;
    cout<<"Tabloul sortat: \n";
    for (i=0; i<n; i++) cout<<B[i]<<" ";
    cout << endl;
    return 0;
}
```

Un tip de date *tablou bidimensional* se defineşte cu ajutorul construcției gramaticale:

typedef <*Tip componente*> <*Nume tip tablou*>[<*Număr linii*>][<*Număr coloane*>];

Pentru exemplificare, în *figura 1.2** este prezentată structura datelor tipului `Matrice`:

```
typedef double Matrice[3][4];
```

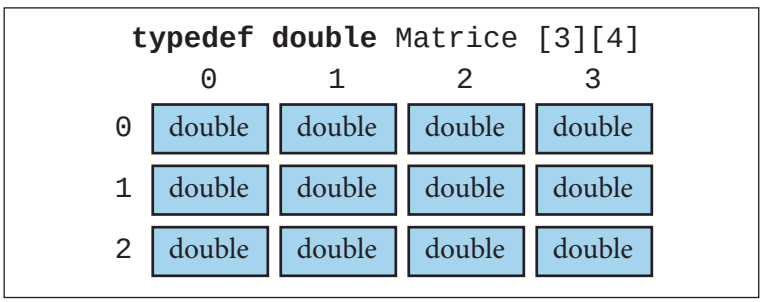


*Fig. 1.2 **. Structura datelor de tip `Matrice`

Componentele unei variabile de tip *tablou bidimensional* se specifică explicit prin numele variabilei urmat de indicii respectivi încadrați de paranteze pătrate.

```
Matrice M;
```

notația `M[1,1]` specifică componenta din linia 1, coloana 1 (*fig. 1.2**); notația `M[1,2]` specifică componenta din linia 1, coloana 2; notația `M[i,j]` specifică componenta din linia `i`, coloana `j`.

Programul ce urmează afişează pe ecran suma componentelor variabilei `M` de tip `Matrice`. Valorile componentelor `M[0,0]`, `M[0,1]`, ..., `M[2,3]` se citesc de la tastatură.

```
// Program P81
// Suma componentelor variabilei M de tip Matrice
#include<iostream>
using namespace std;
int main()
{
  typedef double Matrice[3][4];
  Matrice M;
  int i, j;
  double S;
  cout << "Dati componentele M[i,j]: " << endl;
  for (i=0; i<3; i++)
    for (j=0; j<4; j++)
      {
         cout << "M[" <<i<< ',' << j << "]= ";
         cin >> M[i][j];
      }
  cout << "Ati introdus:" << endl;
  for (i=0; i<3; i++)
    {
      for (j=0; j<4; j++) cout << M[i][j] << ' ';
      cout << endl;
    }
  S=0;
  for (i=0; i<3; i++)
    for (j=0; j<4; j++)
       S=S+M[i][j];
  cout << "Suma= " << S << endl;
  return 0;
}
```

În momentul declarării, elementele tabloului pot fi inițializate, de exemplu:

```
int T[4]={2,5,9,-4};
```

```
int M[3][2]={{9,7}, {-12,8}, {0,-4}};
```

Accentuăm faptul că în limbajul C++ indicii componentelor tabloului pot lua doar valori consecutive întregi, începând cu zero: 0, 1, 2, …, <*Număr componente*> –1. Uneori, acest lucru poate crea anumite incomodități.

Astfel, în cazul exemplului referitor la consumul zilnic de energie electrică pe durata unei luni (vezi paragraful precedent), tabelul respectiv poate fi reprezentat pe calculator cu ajutorul următorului tip de date *tablou unidimensional*:

```
typedef float Consum[31];
Consum C;
```

Indicele tabloului `C` poate lua doar valorile 0, 1, 2, … 30, iar componentele tabloului se specifică prin `C[0]`, `C[1]`, …, `C[30]`. Evident, ultimei zile a lunii,

a 31-a, nu-i va corespunde nicio componentă a tabloului, iar tentativa de a utiliza specificarea `C[31]` ar putea declanşa, în anumite situaţii, o eroare de execuţie. Indiscutabil, am putea să numerotăm zilele lunii începând nu cu „1”, ci cu „0”, însă acest lucru poate face programele C++ mai puţin intuitive.

Pentru a evita astfel de situaţii, se recomandă majorarea numărului de componente ale unui tablou cu o unitate faţă de cel cerut de datele structurate ce trebuie reprezentate pe calculator.

De exemplu, în cazul consumului zilnic de energie electrică pe durata unei luni, putem utiliza următorul tip de date:

```
typedef float Consum[32];
Consum C;
```

în care primei zile a lunii îi va corespunde componenta `C[1]`, zilei a doua — componenta `C[2]`, zilei a treia — componenta `C[3]` ş.a.m.d., iar componenta C[0] va rămâne neutilizată.

Într-un mod similar, pentru reprezentarea pe calculator a consumului zilnic de energie electrică pe parcursul unui an, poate fi folosit următorul tip de date:

```
typedef float Consum[13][32];
Consum C;
```

în care zilelor lunii *Ianuarie* le vor corespunde componentele `C[1, 1]`, ..., `C[1, 31]`; zilelor lunii *Februarie* — componentele `C[2, 1]`, ..., `C[2, 31]`, zilelor lunii *Decembrie* — componentele `C[12, 1]`, ..., `C[12, 31]`.

Evident, unele componente ale tabloului `C`, în special cele din rândul şi coloana 0, vor rămâne neutilizate. Însă întrucât lizibilitatea şi corectitudinea programelor sunt cu mult mai importante decât neutilizarea anumitor componente, majoritatea informaticienilor consideră că acest fapt nu înrăutăţeşte calitatea programelor C++.

Întrebări şi exerciţii

❶ OBSERVĂ! Precizaţi tipul indicilor şi tipul componentelor din următoarele declaraţii:

PASCAL

```
type
P = array [1..5] of integer;
Culoare = (Galben, Verde,
Albastru, Violet);
R = array [Culoare] of real;
S = array [Culoare, 1..3]
of boolean;
T = array [boolean]
of Culoare;
```

C++

```
typedef int P[5];
enum Culoare {Galben, Verde,
     Albastru, Violet};
typedef Culoare R[3];
typedef bool S[4][3];
typedef double T[2][5];
```

Reprezentaţi structura datelor de tipul `P`, `R`, `S` şi `T` pe un desen (vezi figurile respective de mai sus).

❷ (PASCAL) Indicaţi pe diagrama sintactică din *figura 1.3* drumurile care corespund declaraţiilor din exerciţiul 1.

❸ (PASCAL) Scrieți formulele metalingvistice care corespund diagramei sintactice *<Tip tablou>* din *figura 1.3.*

❹ ANALIZEAZĂ! Identificați care dintre următoarele declarații sunt corecte:

PASCAL	C++
`a: array[1..30] of integer;`	`int a[30];`
`a: array[1..10] of byte;`	`char a[1..10];`
`a: array[1.10..10.00] of real;`	`float a[1.10..10.00];`
`a: array[1..50] of char;`	`char a[50];`

❺ EXERSEAZĂ! Se consideră declarațiile:

PASCAL	C++
`type Vector = array [1..5] of real;` `var x, y : Vector;`	`typedef double Vector[5];` `Vector x,y;`

Scrieți expresia aritmetică a cărei valoare este:
a) suma primelor trei componente ale variabilei `x`;
b) suma tuturor componentelor variabilei `y`;
c) produsul tuturor componentelor variabilei `x`;
d) valoarea absolută a componentei a treia a variabilei `y`;
e) suma primelor componente ale variabilelor `x` și `y`.

❻ ANALIZEAZĂ! Pentru următoarele declarații și instrucțiuni:

Declarație	Instrucțiune	Rezultat
PASCAL		
1) `var a:array [1..5] of byte;`	`for i:=1 to 5 do a[i]:=i-1;`	a) 1 1 1 1 1 b) 0 0 0 0 0 c) 0 1 2 3 4 d) 1 2 3 4 0
2) `var v:array[0..4] of integer;`	`for i:=1 to 5 do v[i]:=2*(i-1);`	a) 2 4 6 8 1 b) 0 2 4 6 8 c) 0 1 2 3 4 d) 1 2 3 4 5

C++		
1) `unsigned char a[4];`	`for (i=0; i<5; i++) a[i]=i;`	a) 1 1 1 1 1 b) 0 0 0 0 0 c) 0 1 2 3 4 d) 1 2 3 4 0
2) `int v[4];`	`for (i=0; i<5; i++) v[i]=2*i;`	a) 2 4 6 8 10 b) 0 2 4 6 8 c) 0 1 2 3 4 d) 1 2 3 4 5

Selectați din lista din partea dreaptă variantele ce conțin rezultatele corecte ale execuției fiecăreia dintre aceste instrucțiuni.

❼ REZOLVĂ! Se consideră declarațiile:

PASCAL

```
type Zi = (L, Ma, Mi, J, Vi, S, D);
     Venit = array [Zi] of real;
var v : Venit;
```

C++

```
typedef double Venit[7];
Venit v;
```

Componentele variabilei v reprezintă venitul zilnic al unei întreprinderi. Elaborați un program care:
a) calculează venitul săptămânal al întreprinderii;
b) calculează media venitului zilnic;
c) indică ziua în care s-a obținut cel mai mare venit;
d) indică ziua cu venitul cel mai mic.

❽ REZOLVĂ! Se consideră declarațiile:

PASCAL

```
type Ora = 0..23;
     Grade = -40..40;
     Temperatura = array [Ora]
of Grade;
var t : Temperatura;
```

C++

```
typedef int Ora
typedef int Temperatura [24];
Temperatura t;
```

Componentele variabilei t reprezintă temperaturile măsurate din oră în oră pe parcursul a 24 de ore. Elaborați un program care:
a) calculează temperatura medie;
b) indică maximul şi minimul temperaturii;
c) indică ora (orele) la care s-a înregistrat temperatura maximă;
d) indică ora (orele) la care s-a înregistrat temperatura minimă;
e) calculează numărul de zile în care au fost înregistrate temperaturi mai jos de zero grade;
f) calculează numărul de zile în care au fost înregistrate temperaturi mai mari de media săptămânală.

❾ Se consideră declarațiile:

PASCAL

```
type Oras = (Chisinau, Orhei,
Balti, Tighina, Tiraspol);
    Zi=(L, Ma, Mi, J, Vi, S, D);
 Consum = array [Oras,Zi] of real;
var C : Consum;
    r : Oras;
    z: Zi;
```

C++

```
enum Oras {Chisinau, Orhei,
    Balti, Tighina, Tiraspol};
enum Zi {L, Ma, Mi, J, Vi, S, D};
typedef double Consum[5][7];
Consum C;
Oras r;
Zi z;
```

Componenta C[r, z] / C[r][z] în C++ a variabilei C reprezintă consumul de energie electrică a oraşului r în ziua z. Elaborați un program care:
a) calculează energia electrică consumată de fiecare oraş pe parcursul unei săptămâni;
b) calculează energia electrică consumată zilnic de oraşele în studiu;
c) indică oraşul cu un consum săptămânal maxim;
d) indică oraşul cu un consum săptămânal minim;
e) indică ziua în care oraşele consumă cea mai multă energie electrică;
f) indică ziua în care oraşele consumă cea mai puțină energie electrică.

❿ Se consideră declarațiile

PASCAL

```
type Vector = array [1..5] of real;
Matrice = array [1..3, 1..4] of
boolean;
Linie = array [1..4] of real;
Tabel = array [1..3] of Linie;
var V : Vector;
    M : Matrice;
    L : Linie;
    T : Tabel;
    x : real;
    i : integer;
```

C++

```
typedef float Vector[5];
typedef bool Matrice[3][4];
typedef float Linie[4];
typedef Linie Tabel[3];
Vector V;
Matrice M;
Linie L;
Tabel T;
float x;
int i;
```

Care dintre atribuirile ce urmează sunt corecte?

	PASCAL		**C++**
a)	`T[3]:=T[1]`	*a)*	`T[3]=T[1]`
b)	`M:=T`	*b)*	`M=T`
c)	`L:=V`	*c)*	`L=V`
d)	`L[3]:=x`	*d)*	`L[3]=x`
e)	`x:=i`	*e)*	`x=i`
f)	`i:=x`	*f)*	`i=x`
g)	`L[3]:=i`	*g)*	`L[3]=i`
h)	`i:=M[1,2]`	*h)*	`i=M[1][2]`
i)	`x:=V[4]`	*i)*	`x=V[4]`
j)	`L[3]:=V[4]`	*j)*	`L[3]=V[4]`
k)	`T[1]:=4`	*k)*	`T[1]=4`
l)	`T[2]:=V`	*l)*	`T[2]=V`
m)	`L:=T[3]`	*m)*	`L=T[3]`
n)	`T[1,2]:=M[1,2]`	*n)*	`T[1][2]=M[1][2]`
o)	`T[2,1]:=M[1,2]`	*o)*	`T[2][1]=M[1][2]`
p)	`M[1]:=4`	*p)*	`M[1]=4`
q)	`M[1,3]:=L[2]`	*q)*	`M[1][3]=L[2]`
r)	`x:=T[1][2]`	*r)*	`x=T[1,2]`
s)	`x:=M[1]`	*s)*	`x=M[1]`
t)	`L:=M[1]`	*t)*	`L=M[1]`
u)	`V[5]:=M[3,4]`	*u)*	`V[5]=M[3][4]`
v)	`L:=M[3,4]`	*v)*	`L=M[3][4]`

⓫ REZOLVĂ! De la tastatură se citesc elementele unui tablou unidimensional, format din n numere întregi, $n<10$. Elaborați un program care:
a) afişează pe ecran componentele tabloului la un interval de 5 poziții;
b) afişează pe ecran numerele în ordinea inversă a introducerii în calculator;
c) sortează componentele tabloului în ordine descrescătoare;
d) afişează pe ecran doar componentele pare;
e) afişează pe ecran media aritmetică a componentelor pare;
f) afişează pe ecran doar componentele impare;
g) afişează pe ecran doar componentele care sunt mai mari ca x şi nu sunt divizibile cu y (valorile x şi y se citesc de la tastatură);
h) afişează pe ecran doar componentele care sunt mai mari ca x şi mai mici decât y (valorile x şi y se citesc de la tastatură);
i) afişează pe ecran pozițiile (indicii) componentelor impare negative;
j) afişează pe ecran pozițiile (indicii) componentelor ce conțin doar două cifre semnificative;
k) înlocuieşte prima componentă a tabloului cu componenta de valoare minimă din tabloul respectiv;
l) înlocuieşte componenta de valoare minimă din tabloul respectiv cu prima componentă a acestuia;
m) creează un tablou nou, format doar din componentele pare ale tabloului introdus de la tastatură;
n) creează un tablou nou, format doar din componentele divizibile cu 3 ale tabloului introdus de la tastatură;
o) creează un tablou nou, format doar din acele componente ale tabloului introdus de la tastatură care au cel mult patru divizori.

⓬ REZOLVĂ! O persoană intenționează să cumpere n produse distincte, numerotate prin i =1, 2, 3, …, n. Fiecare dintre aceste produse poate fi procurat în oricare dintre cele k magazine disponibile, notate prin j = 1, 2, 3, …, k. Evident, cumpărătorul optează pentru magazinele cu cele mai mici prețuri. Elaborați un program care determină pentru fiecare produs i magazinul j în care el trebuie cumpărat şi costul total C al cumpărăturilor făcute în acest mod.
Date de intrare: tabloul bidimensional P, în care componenta $P[i][j]$ reprezintă prețul produsului i în magazinul j.
Date de ieşire: costul total al cumpărăturilor C şi tabloul unidimensional M, în care componenta $M[i]$ reprezintă magazinul j în care va fi procurat produsul i.
Restricții: $1 \le n \le 10$. $1 \le k \le 5$. Prețurile sunt indicate prin numere reale, cu două cifre după punctul zecimal.

⓭ STUDIU DE CAZ. Proiectați structurile de date de tip tablou necesare pentru reprezentarea în programele PASCAL/C++ a tabelelor ce conțin informațiile referitoare la consumul zilnic de energie electrică, respectiv, pe durata unei luni şi pe durata unui an, descrise în paragraful precedent. Analizați cum numărul de componente din declarațiile de tablouri influențează înțelegerea programelor de către utilizatori.

⓮ ÎNVAȚĂ SĂ ÎNVEȚI! Tablourile bidimensionale cu acelaşi număr de linii şi coloane se numesc *tablouri pătratice*. Mulțimea componentelor care au indicele de linie egal cu indicele de coloană formează *diagonala principală* a unui tablou pătratic. Evident, această mulțime poate fi simbolizată printr-o linie imaginară ce reuneşte componenta din colțul stânga-sus cu componenta din colțul dreapta-jos al tabloului pătratic. Într-un mod similar, poate fi definită şi diagonala secundară a tabloului pătratic, care, de asemenea, poate fi simbolizată printr-o

linie imaginară ce reunește componenta din colțul stânga-jos cu componenta din colțul dreapta-sus.
Elaborați un program care citește de la tastatură un tablou pătratic cu n linii, $2 \leq n \leq 10$, și afișează la ecran suma componentelor care se află:
a) pe diagonala principală;
b) pe diagonala secundară;
c) mai sus de diagonala principală;
d) mai jos de diagonala principală;
e) mai sus de diagonala secundară;
f) mai jos de diagonala secundară.
Se consideră că componentele tabloului pătratic sunt numere întregi, care se citesc de la tastatură.

⓯ DESCOPERĂ! Limbajul C ++ oferă un tip alternativ de tablouri bidimensionale, definit în biblioteca `<array>`. Explorați această bibliotecă și rescrieți programele din acest paragraf folosind oportunitățile oferite de ea. Prin ce se aseamănă și prin ce diferă aceste două moduri de definire a tablourilor?

1.3. Tipuri de date *șir de caractere*

Metodele de reprezentare și de prelucrare a șirurilor de caractere au o importanță deosebită, întrucât foarte multe obiecte din lumea reală sunt descrise nu doar prin numere, ci și prin cele mai diverse texte. De obicei, în mediile moderne de dezvoltare a programelor, șirurile de caractere pot fi reprezentate prin mai multe structuri de date. În continuare vom studia următoarele tipuri de date utilizate pentru reprezentarea șirurilor de caractere:
– tablourile unidimensionale cu componente de tipul `char` (caractere);
– *string*-urile.
Amintim că în cazul limbajelor de programare, cuvântul englez *string* are semnificația de șir, serie, succesiune de caractere.

PASCAL

Șiruri de caractere de tipul *tablouri unidimensionale*

În limbajul-standard tipul de date *șir de caractere* reprezintă un caz special al tipului **`array`** și se definește printr-o construcție de forma

<*Nume tip*> ::= **`packed array`** `[1..n]` **`of`** `char;`

Mulțimea de valori ale tipului de date în studiu este formată din toate șirurile ce conțin exact n caractere.

Exemplu:

```
Program P82;
{ Siruri de caractere de tip tablou unidimensional }
type     Nume = packed array [1..8] of char;
         Prenume = packed array [1..5] of char;
var  N : Nume;
     P : Prenume;
```

```
begin
  N:='Munteanu';
  P:='Mihai';
  writeln(N);
  writeln(P);
  readln;
end.
```

Rezultatul afişat pe ecran:

```
Munteanu
Mihai
```

Întrucât şirurile de lungime diferită aparţin unor tipuri distincte de date, în cadrul programului P82 nu sunt admise atribuiri de genul:

```
N:= 'Olaru';
P:= 'Ion'.
```

În astfel de cazuri, programatorul va completa datele respective cu spaţiu până la numărul stabilit de caractere n, de exemplu:

```
N:= 'Olaru ';
P:= 'Ion '.
```

Valorile unei variabile v de tip **packed array** [1..n] **of** char pot fi introduse de la tastatură numai prin citirea separată a componentelor respective:

```
read(v[1]); read(v[2]); ...; read(v[n]).
```

În schimb, o astfel de valoare poate fi afişată în totalitatea ei printr-un singur apel write(v) sau writeln(v).

Accentuăm faptul că şirurile de caractere de tip **packed array** [1..n] **of** char conţin exact n caractere, adică sunt **şiruri de lungime constantă**. Evident, lungimea lor nu poate fi modificată pe parcursul derulării programului respectiv. Acest fapt complică elaborarea programelor destinate prelucrării unor şiruri arbitrare de caractere.

Şiruri de caractere de tipul string

Pentru a uşura munca programatorilor, versiunile actuale ale limbajului PASCAL permit utilizarea tipului de date **string**, mulţimea de valori a căruia este formată din **şiruri de caractere de lungime variabilă**. Un astfel de tip se declară printr-o construcţie de forma:

type <*Nume tip*> = **string**; sau **type** <*Nume tip*> = **string** [*nmax*];

unde *nmax* este lungimea maximă pe care o pot avea şirurile respective. În lipsa parametrului *nmax* lungimea maximă se stabileşte implicit, în mod obişnuit – 255 de caractere.

Asupra şirurilor de tip **string** se poate efectua operaţia de concatenare (juxtapunere), notată prin semnul „+”. Lungimea curentă a unei valori v de tip **string** poate fi aflată cu ajutorul funcţiei predefinite length(v) care returnează

o valoare de tip `integer`. Indiferent de lungime, toate şirurile de caractere de tip **`string`** sunt compatibile.

Exemplu:

```
Program P83_a;
{ Siruri de caractere de tip string }
type Nume = string [8];
     Prenume = string [5];
     NumePrenume = string;
var N : Nume;
    P : Prenume;
    NP : NumePrenume;
    L : integer;
begin
  N:='Munteanu';   L:=length(N);    writeln(N, L:4);
  P:='Mihai';      L:=length(P);    writeln(P, L:4);
  NP:=N+' '+P;     L:=length(NP);   writeln(NP, L:4);
  N:='Olaru';      L:=length(N);    writeln(N, L:4);
  P:='Ion';        L:=length(P);    writeln(P, L:4);
  NP:=N+' '+P;     L:=length(NP);   writeln(NP, L:4);
  readln;
end.
```

Rezultatele afişate pe ecran:

```
Munteanu   8
Mihai   5
Munteanu Mihai  14
Olaru   5
Ion   3
Olaru Ion 9
```

Se observă că pe parcursul derulării programului în studiu lungimea şirurilor de caractere `N`, `P` şi `NP` se schimbă.

Asupra şirurilor de caractere sunt admise operaţiile relaţionale <, <=, =, >=, >, <>. Şirurile se compară componentă cu componentă, de la stânga la dreapta, în conformitate cu ordonarea caracterelor în tipul de date `char`. Ambii operanzi trebuie să fie de tip **`packed array`** `[1..n]` **`of`** `char` cu acelaşi număr de componente sau de tip **`string`**. Evident, operanzii de tip **`string`** pot avea lungimi arbitrare.

De exemplu, rezultatul operaţiei

```
'AC' < 'BA'
```

este `true`, iar rezultatul operaţiei

```
'AAAAC' < 'AAAAB'
```

este `false`.

O variabilă de tip *şir de caractere* poate fi folosită fie în totalitatea ei, fie parţial, prin referirea unui caracter din şir.

De exemplu, pentru `P='Mihai'` avem `P[1]='M'`, `P[2]='i'`, `P[3]='h'` ş.a.m.d. După executarea secvenței de instrucțiuni

```
P[1]:='P';
P[2]:='e';
P[3]:='t';
P[4]:='r';
P[5]:='u'
```

variabila P va avea valoarea `'Petru'`.

Programul ce urmează citeşte de la tastatură şiruri arbitrare de caractere şi afişează pe ecran numărul de spații în şirul respectiv. Derularea programului se termină după introducerea şirului `'Sfarsit'`.

```
Program P83_b;
{ Numarul de spatii într-un sir de caractere }
var S : string;
    i, j : integer;
begin
  writeln('Dati siruri de caractere:');
  repeat
    readln(S);
    i:=0;
    for j:=1 to length(S) do
      if S[j]=' ' then i:=i+1;
    writeln('Numarul de spatii=', i);
  until S='Sfarsit';
end.
```

Menționăm faptul că compilatoarele moderne ale limbajului PASCAL conțin mai multe funcții standard de prelucrare a şirurilor de caractere, de exemplu, concatenarea, copierea, secvențierea, inserarea sau ştergerea anumitor caractere din şir ş.a.m.d. Descrierea acestor funcții poate fi găsită în sistemele de asistență ale mediilor integrate de dezvoltare a programelor.

C++

Şiruri de caractere de tipul *tablouri unidimensionale*

În programele C++ şirurile de caractere pot fi reprezentate prin *tablouri unidimensionale*, componentele cărora sunt de tip **char**. Astfel de tipuri se declară cu ajutorul construcției gramaticale de forma:

typedef char <*Nume tip şir*> `[nmax];`

unde `nmax` indică numărul maxim de caractere pe care îl pot avea şirurile de tipul respectiv.

Mulțimea de valori ale tipului de date <*Nume tip şir*> este formată din şiruri ce conțin până la `nmax` caractere. Întrucât în reprezentarea internă una dintre componentele tabloului unidimensional ce reprezintă şirul de caractere se utilizează pentru a indica sfârşitul şirului, lungimea şirului va fi de cel mult `nmax-1` caractere semnificative.

Exemple:

1)
```
typedef char Nume[20];
Nume N;
```

2)
```
typedef char Prenume[10];
Prenume P;
```

3)
```
typedef char Companie[45];
Companie C;
```

După cum a fost menționat mai sus, în reprezentarea internă, sfârşitul de şir se indică cu ajutorul unui simbol special, şi anume, simbolul *escape* „\0".

De exemplu, dacă variabila P de tip Prenume conține şirul de caractere „Ion", tabloul respectiv are forma:

0	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*
I	o	n	\0						

Dacă însă aceeaşi variabilă P va conține şirul de caractere "Cristina", tabloul respectiv va avea forma:

0	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*
C	r	i	s	t	i	n	a	\0	

Prin urmare, în pofida faptului că în declarația tipului de date Prenume sunt indicate 10 caractere, şirurile de acest tip pot conține până la 9 caractere semnificative.

În principiu, şirurile de caractere reprezentate prin tablouri unidimensionale pot fi prelucrate accesând componentele dorite ale acestora.

De exemplu, dacă se doreşte înscrierea în variabila P a şirului de caractere "Ion", în acest scop pot fi folosite următoarele instrucțiuni:

```
P[0]= 'I';
P[1]= 'o';
P[2]= 'n';
P[3]= '\0';
```

Accentuăm faptul că indicarea sfârşitului de şir cu ajutorul simbolului *escape* „\0" este obligatorie. În caz contrar, vor apărea erori de execuție.

Valorile variabilelor de tip şiruri de caractere – tablouri unidimensionale, pot fi afişate pe ecran cu ajutorul operatorului de transfer al datelor << în fluxul de ieşire cout.

Exemplu:

```
// Program P82_a
// Siruri de caractere de tip tablou unidimensional
#include <iostream>
using namespace std;
int main()
{
```

```
    typedef char Nume[20];
    Nume N;
    typedef char Prenume[10];
    Prenume P;
    N[0]='L'; N[1]='u'; N[2]='p'; N[3]='u'; N[4] = '\0';
    P[0]='I'; P[1]='o'; P[2]='n'; P[3]='\0';
    cout << N << endl;
    cout << P << endl;
    return 0;
}
```

Rezultatele afişate pe ecran:

```
Lupu
Ion
```

La **citirea şirurilor de caractere** de la tastatură se va ține cont de faptul că intrarea standard este privită ca o sursă ce furnizează fluxul încontinuu de caractere `cin`. Pentru a putea extrage din acest flux şirurile dorite, programul trebuie să cunoască care caracter le delimitează. De obicei, în calitate de *delimitator* se folosesc aşa-numitele *caractere albe* (spațiul, tabulatorul →|, sfârşitul de linie), însă programatorul poate utiliza şi alte caractere.

Pentru citirea şirurilor de caractere **delimitate prin caractere albe** se foloseşte operatorul >>. Pentru afişarea şirurilor de caractere se foloseşte operatorul <<.

Exemplu:

```
// Program P82_b
// Citirea sirurilor de caractere
// Delimitatori: spatiu sau sfarsit de linie
#include <iostream>
using namespace std;
int main()
{
  typedef char Nume[20];
  Nume N;
  typedef char Prenume[10];
  Prenume P;
  cin >> N;  // citirea sirului N
  cin >> P;  // citirea sirului P
  cout << N << endl;  // afisarea sirului N
  cout << P << endl;  // afisarea sirului P
  return 0;
}
```

Presupunem că după lansarea programului în execuție, utilizatorul tastează:

```
Margine<ENTER>
Cristina<ENTER>
```

În acest caz, fluxul de intrare `cin` va avea forma:

```
Margine<ENTER>Cristina<ENTER>
↑
```

unde simbolul ↑ reprezintă cursorul acestuia. Inițial, cursorul se află la începutul fluxului, adică în fața caracterului `M`.

În procesul executării instrucțiunii `cin>>N`, programul, începând cu poziția curentă a cursorului, parcurge fluxul de intrare de la stânga la dreapta, mutând cursorul caracter după caracter şi depunând caracterele citite în variabila `N`. Imediat ce cursorul ajunge la delimitatorul sfârşit de linie *<ENTER>*, instrucțiunea `cin>>N` înscrie în variabila `N` caracterul *escape* `'\0'` şi termină citirea primului şir de caractere. Evident, după executarea instrucțiunii `cin>>N`, variabila `N` va primi valoarea `"Margine"`, iar cursorul se va afla în fața caracterului `C`.

Într-un mod similar, instrucțiunea `cin>>P` continuă parcurgerea fluxului de intrare, mutând cursorul şi depunând caracterele citite, unul după altul, în variabila `P`. Imediat ce cursorul ajunge la delimitatorul sfârşit de linie *<ENTER>*, instrucțiunea `cin>>P` înscrie în variabila `P` caracterul *escape* `'\0'` şi termină citirea celui de al doilea şir de caractere. Evident, după executarea instrucțiunii `cin>>P`, variabila `P` va primi valoarea `"Cristina"`, iar cursorul se va afla la sfârşitul fluxului.

Prin urmare, programul va afişa pe ecran:

```
Margine
Cristina
```

Accentuăm faptul că instrucțiunile de citire a datelor din fluxul de intrare `cin >>` folosesc în calitate de delimitator nu doar sfârşitul de linie *<ENTER>*, dar şi spațiul. Astfel, dacă utilizatorul va tasta:

```
Margine Cristina Lupu Ion<ENTER>
                 ↑
```

variabila `N` va primi valoarea `"Margine"`, variabila `P` – valoarea `"Cristina"`, iar cursorul se va afla în fața caracterului `L`. Restul caracterelor din fluxul de date, şi anume `"Lupu Ion"`, vor rămâne stocate în fluxul de intrare.

Pentru a citi şirurile de caractere **delimitate doar de sfârşit de linie**, adică de acționarea tastei *<ENTER>*, se foloseşte funcția `cin.getline`. Apelul acestei funcții are forma:

`cin.getline` (*S*, *n*);

unde:

S – variabila în care va fi depus şirul de caractere extras din fluxul de intrare;

n – numărul maxim de caractere ce pot fi citite din fluxul de intrare.

Exemplu:

```
// Program P82_c
// Citirea sirurilor de caractere
// Delimitator: sfarsitul de linie
#include <iostream>
```

```
using namespace std;
int main()
{
  typedef char NumePrenume[30];
  NumePrenume NP;
  cin.getline(NP, 30);  // citirea sirului NP
  cout << NP << endl;
  return 0;
}
```

Dacă, în procesul executării programului de mai sus, utilizatorul va tasta:

```
Paduraru Elena - Promovata<ENTER>
```

variabila NP va primi valoarea "Paduraru Elena - Promovata".

O altă modalitate de citire a unui şir care poate conține spații este folosirea funcției get().

Întrucât prelucrarea şirurilor de caractere prin accesarea fiecăreia dintre componentele tablourilor respective este anevoioasă, limbajul C++ conține un şir de funcții predefinite care simplifică acest proces. Prezentăm în continuare câteva dintre ele:

strcpy(S_1, S_2) – copie şirul de caractere S_2 în şirul S_1;
strcat(S_1, S_2) – alipeşte la şirul S_1 şirul de caractere S_2;
strlen(S) – returnează lungimea şirului de caractere S.

Mediile de dezvoltare ale programelor C++ conțin şi alte funcții predefinite, destinate prelucrării şirurilor de caractere. Descrierea şi modul de utilizare a acestor funcții pot fi găsite în sistemele de asistență ale mediilor respective.

Pentru a putea utiliza aceste funcții, în programul în curs de elaborare trebuie inclusă directiva

```
#include <cstring>;
```

Exemplu:

```
// Program P82_d
/* Functii predefinite pentru prelucrarea */
/* sirurilor de tip tablou unidimensional */
#include <iostream>
#include <cstring>
using namespace std;
int main()
{
  typedef char Nume[20];
  Nume N;
  typedef char Prenume[10];
  Prenume P;
  strcpy(N, "Munteanu");
  cout << N << "   " << strlen(N) << endl;
```

```
  strcpy(P, "Mihai");
  cout << P << "   " << strlen(P) << endl;
  strcat(N, " ");
  cout << strcat(N, P) << "   " << strlen(N) << endl;
  return 0;
}
```

Rezultatele afişate pe ecran:

```
Munteanu   8
Mihai   5
Munteanu Mihai   14
```

Evident, în cazul reprezentării şirurilor de caractere prin tablouri unidimensionale, ele pot fi iniţializate în momentul declarării. Se va ţine însă cont de faptul că tablourile respective trebuie să conţină în mod obligatoriu caracterul *escape* `'\0'` (sfârşit de şir).

Exemple:

1)
```
char S[]={'I', 'o', 'n', '\0'};
```

2)
```
char S[]="Ion";
```

3)
```
char T[]={'E', 'l', 'e', 'n', 'a', '\0'};
```

Şiruri de caractere de tipul string

Cunoaştem deja că şirurile de caractere de tip **char** <*Nume tip şir*> `[nmax]` pot conţine cel mult `nmax-1` caractere semnificative. Însă, foarte des, la momentul scrierii programelor C++, valoarea `nmax` este necunoscută, fapt ce complică procesul de programare.

Pentru a uşura munca programatorilor, limbajul de programare C++ conţine tipul predefinit de date `string`, care permite declararea şirurilor de caractere fără a indica lungimea maximă posibilă a acestora. Pentru a utiliza acest tip, în partea declarativă a programului C++ trebuie inserată directiva:

```
#include <string>;
```

Şirurile de caractere de tip `string` pot apărea în instrucţiunile de atribuire şi asupra lor poate fi efectuată operaţia de concatenare (de alipire), notată prin simbolul „+”.

Exemplu:

```
// Program P83_a
// Siruri de caractere de tip string
#include <iostream>
#include <string>
using namespace std;
int main()
{
```

```
    string N;            // Nume
    string P;            // Prenume
    string NP;           // Nume si Prenume
    N = "Olaru";         // atribuire
    P = "Andrei";        // atribuire
    NP = N + " " + P;    // concatenare si atribuire
    cout << N << endl;
    cout << P << endl;
    cout << NP << endl;
    return 0;
}
```

Acest program va afişa pe ecran:

```
Olaru
Andrei
Olaru Andrei
```

Asupra şirurilor de caractere sunt admise următoarele operaţii relaţionale: <, <=, ==, >=, >, !=. Şirurile se compară componentă cu componentă, de la stânga la dreapta, în conformitate cu ordonarea caracterelor în tipul de date `char`.

De exemplu, rezultatul operaţiei

```
'AC' < 'BA'
```

este `true`, iar rezultatul operaţiei

```
'AAAAC' < 'AAAAB'
```

este `false`.

O variabilă de tip şir de caractere poate fi folosită fie în totalitatea ei, fie parţial, prin referirea unui caracter din şir.

De exemplu, după executarea secvenţei de instrucţiuni

```
P="Mihai";
P[0]='P'; P[1]='e'; P[2]='t'; P[3]='r'; P[4]='u';
```

variabila `P` va avea valoarea `"Petru"`.

De obicei, citirea unei linii introduse de la tastatură într-o variabilă *S* de tip `string` se efectuează cu ajutorul unui apel de forma:

```
getline(cin, S);
```

Lungimea *l* a unui şir de caractere *S* de tip `string` poate fi aflată cu ajutorul unei instrucţiuni de forma:

```
l = S.length();
```

unde *S* este şirul de caractere supus prelucrării.

Programul ce urmează citeşte de la tastatură şiruri arbitrare de caractere şi afişează pe ecran numărul de spaţii în fiecare şir citit. Derularea programului se termină după introducerea şirului `'Sfarsit'`.

```
// Program P83_b
// Numarul de spatii intr-un sir
#include <iostream>
#include <string>
using namespace std;
int main()
{
    string Sir; // Sirul citit de la tastatura
    int L;      // Lungimea sirului citit
    int n;      // Numarul de spatii in sirul citit
    int i;
Repeta:
    cout << "Dati sirul de caractere:" << endl;
    getline(cin, Sir);  // Citirea sirului
    L = Sir.length();   // Lungimea sirului
    cout << "Lungimea sirului = " << L << endl;
    n = 0;
    for (i=0; i < L; i++)
      if (Sir[i] == ' ') n++;
    cout << "Numarul de spatii in sir = " << n << endl;
    cout << endl;
    if (Sir != "Sfarsit") goto Repeta;
    return 0;
}
```

Bibliotecile standard ale limbajului C++ oferă programatorului posibilitatea să utilizeze ambele reprezentări ale şirurilor de caractere, atât prin tablouri, cât şi prin *string*-uri.

Transformarea din *string* în tablou se poate face cu ajutorul unui apel de forma:

```
S.c_str();
```

unde *S* este o variabilă de tip *string*.

Transformarea din tablou de caractere în *string* se poate face printr-o simplă atribuire de forma *S* = *T*.

Exemplu:

```
// Program P83_c
// Transformari string <--> tablou de caractere
#include <iostream>
#include <cstring>
using namespace std;
int main()
{
  char A[20];  // A este un tablou de caractere
  string B;    // B este un string
  // Transformarea string-ului B in tabloul A
  B="Informatica";
```

```
    strcpy(A, B.c_str());
    cout << A << endl;
    // Transformarea tabloului A in string-ul B
   A[0]='A'; A[1]='d'; A[2]='m'; A[3]='i'; A[4]='s'; A[5]='\0';
    B=A;
    cout << B << endl;
    return 0;
  }
```

Întrebări şi exerciţii

❶ Cum se defineşte un tip de date *şir de caractere*? Care sunt cele două modalităţi de reprezentare a şirurilor de caractere în programele PASCAL şi C++?

❷ Ce operaţii pot fi efectuate asupra şirurilor de caractere reprezentate prin tablouri unidimensionale? Asupra şirurilor de caractere de tipul *string*?

❸ PROGRAMEAZĂ! Elaboraţi un program care propune elevului să introducă de la tastatură, în două linii:
– numele şi prenumele elevului/elevei;
– profesia la care el/ea aspiră.
Folosiţi mesaje sugestive şi organizaţi dialogul om–calculator conform următorului model:

```
Ce nume si prenume ai?
Paduraru Elena
Ce iti doresti sa devii in viitor?
medic
Te numesti Paduraru Elena
Doresti sa devii medic
```

În acest model datele introduse de utilizator sunt evidenţiate prin îngroşare.

❹ OBSERVĂ! Comentaţi următorul program:

PASCAL

```
Program P84;
{ Eroare }
var S : packed array [1..5] of
char;
begin
  S:='12345';
  writeln(S);
  S:='Sfat';
  writeln(S);
end.
```

C++

```
//Program P84
// Eroare
#include <iostream>
using namespace std;
int main()
{
  char S[]="12345";
  cout<<S<<endl;
  S="Sfat";
  cout<<S;
  return 0;
}
```

❺ EXPERIMENTEAZĂ! Inserați înaintea instrucțiunii „`return 0;`” din programul `P82_b` secvența de instrucțiuni:

```
cin >> N;
cin >> P;
cout << N << endl;
cout << P << endl;
```

Lansați în execuție programul modificat și tastați:

```
Margine Cristina Lupu Ion<ENTER>
```

Explicați mesajele afișate pe ecran.

❻ ANALIZEAZĂ! Se consideră șiruri de caractere de tip `string`. Precizați rezultatul operațiilor relaționale:

	PASCAL		C++
a)	`'B' < 'A'`	*a)*	`'B' < 'A'`
b)	`'BB' > 'AA'`	*b)*	`"BB" > "AA"`
c)	`'BAAAA' < 'AAAAA'`	*c)*	`"BAAAA" < "AAAAA"`
d)	`'CCCCD' > 'CCCCA'`	*d)*	`"CCCCD" > "CCCCA"`
e)	`'A' = 'AA'`	*e)*	`"A" == "AA"`
f)	`'BB ' < 'B'`	*f)*	`"BB " < "B B"`
g)	`'A' = 'a'`	*g)*	`"A" == "a"`
h)	`'Aa' > 'aA'`	*h)*	`"Aa" > "aA"`
i)	`'123' = '321'`	*i)*	`"123" == "321"`
j)	`'12345' > '12345'`	*j)*	`"12345" > "12345"`

❼ REZOLVĂ! Elaborați un program care citește de la tastatură șirul de caractere *S* și afișează pe ecran:

a) numărul de apariții ale caracterului `'A'` în șirul *S*;
b) șirul obținut prin substituirea caracterului `'A'` prin caracterul `'*'`;
c) șirul obținut prin radierea din șirul *S* a tuturor aparițiilor caracterului `'B'`;
d) numărul de apariții ale silabei `MA` în șirul *S*;
e) șirul obținut prin substituirea tuturor aparițiilor în șirul *S* a silabei `MA` prin silaba `TA`;
f) șirul obținut prin radierea din șirul *S* a tuturor aparițiilor silabei `TO`;
g) scrierea inversă a șirului *S*;
h) `true` dacă șirul *S* este palindrom și `false` în caz contrar;
i) șirul obținut prin transformarea tuturor literelor mici din componența șirului *S* în litere mari;
j) șirul obținut prin transformarea primei litere a fiecăruia dintre cuvintele din componența șirului *S* în literă mare;
k) șirul obținut prin sortarea în ordine alfabetică a caracterelor din șirul *S*.

❽ PROGRAMEAZĂ! Se consideră șiruri de caractere formate din literele mari ale alfabetului latin și spațiu. Elaborați un program care afișează șirurile în studiu după următoarele reguli:

– fiecare literă de la `'A'` până la `'Y'` se înlocuieşte prin următoarea literă din alfabet;
– fiecare literă `'Z'` se înlocuieşte prin litera `'A'`;
– fiecare spaţiu se înlocuieşte prin `'-'`.

❾ PROGRAMEAZĂ! Elaboraţi un program care descifrează şirurile cifrate conform regulilor din exerciţiul precedent.

❿ PROGRAMEAZĂ! Elaboraţi un program care citeşte de la tastatură şirul S format din n caractere, $n \leq 30$, şi afişează pe ecran toate şirurile formate prin ştergerea din S a ultimelor k caractere, $k = 0, 1, 2, .., n - 1$. De exemplu, pentru S = `Test`, programul va afişa:

```
Test
Tes
Te
T
```

⓫ CREEAZĂ! Elaboraţi un program care:
– citeşte de la tastatură m, $m \leq 100$, şiruri de caractere, formate din literele mici ale alfabetului latin;
– sortează şirurile în cauză în ordine alfabetică;
– afişează şirurile sortate pe ecran.

⓬ PROGRAMEAZĂ! Şirul S este compus din câteva propoziţii. Propoziţiile se termină cu punct, semnul de exclamare sau semnul întrebării. Elaboraţi un program care afişează pe ecran:
– numărul de propoziţii din şirul de caractere introdus de la tastatură;
– numărul de cuvinte din componenţa fiecărei propoziţii.

⓭ ÎNVAŢĂ SĂ ÎNVEŢI! Este cunoscut faptul că compilatoarele moderne conţin mai multe funcţii predefinite destinate prelucrării şirurilor de caractere, de exemplu: concatenarea, copierea, secvenţierea, inserarea sau ştergerea anumitor caractere ş.a.m.d. Descrierea acestor funcţii poate fi găsită în sistemele de asistenţă ale mediilor integrate de dezvoltare a programelor. Studiaţi, în mod individual sau în echipă, funcţiile predefinite de prelucrare a şirurilor de caractere şi completaţi tabelul:

Funcţia	**Descriere**	**Exemple**
...		
...		

Utilizând funcţiile respective, rescrieţi programele din acest paragraf. Estimaţi, în ce măsură utilizarea acestor funcţii simplifică procesul de elaborare a programelor PASCAL/C++.

⓮ EXPLOREAZĂ! În afară de funcţia predefinită `cin.getline`, pentru citirea şirurilor de caractere poate fi folosită şi funcţia `cin.get`. Studiaţi, în mod individual sau în echipă, această funcţie. Aflaţi care sunt deosebirile dintre funcţiile `cin.getline` şi `cin.get`. Determinaţi în care cazuri este indicată folosirea funcţiei `cin.getline` şi in care cazuri – folosirea funcţiei `cin.get`.
Utilizând funcţia `cin.get`, elaboraţi un program care citeşte de la tastatură un şir de caractere şi afişează pe ecran lungimea acestuia. Şirul poate să includă unul sau mai multe spaţii, iar sfârşitul lui este indicat de caracterul `'x'`.

1.4. Tipuri de date *articol*

Articolul, cunoscut în informatică şi sub denumirea de *înregistrare* sau *structură*, reprezintă o grupare de date, reunite sub un singur nume. Articolele sunt formate din componente, denumite ***câmpuri***. Spre deosebire de componentele unui tablou, câmpurile pot fi de tipuri diferite. Fiecare câmp are un nume (identificator de câmp).

De exemplu, rezultatele examenului de bacalaureat pot fi afişate în forma:

	Nume	**Prenume**	**Nota medie**
Nume câmp →			
	Margine	Cristina	10,0
Articol →	Munteanu	Ion	8,7
	...	...	...
	Păduraru	Elena	9,5

În acest model de afişare a rezultatelor examenului de bacalaureat, fiecărui elev îi corespunde un articol (o înregistrare) ce conţine trei câmpuri intitulate *Nume*, *Prenume* şi *Nota medie*. Pe calculator, datele din câmpurile *Nume* şi *Prenume* pot fi reprezentate prin şiruri de caractere, iar cele din câmpul *Nota medie* – prin numere reale.

PASCAL

În limbajul PASCAL, un tip de date *articol* se defineşte printr-o structură gramaticală de forma:

```
type <Nume tip> = record
                      <Nume câmp 1> : T1;
                      <Nume câmp 2> : T2;
                      ...
                      <Nume câmp n> : Tn;
                  end;
```

unde T_1, T_2, .., T_n specifică tipul câmpurilor respective. Tipul unui nume de câmp este arbitrar, astfel un câmp poate să fie, la rândul lui, tot de tip *articol*. Prin urmare, se pot defini tipuri imbricate.

Exemple:

1)
```
type Elev =    record
                  Nume : string;
                  Prenume : string;
                  NotaMedie : real;
               end;
var E1, E2 : Elev;
```

2)
```
type Punct = record
                x : real; { coordonata x }
                y : real; { coordonata y }
             end;
var P1, P2 : Punct;
```

3)
```
type Triunghi = record
                    A : Punct; { vârful A }
                    B : Punct; { vârful B }
                    C : Punct; { vârful C }
                end;
var T1, T2, T3 : Triunghi;
```

Structura datelor din exemplele în studiu este prezentată în *figura 1.4*.

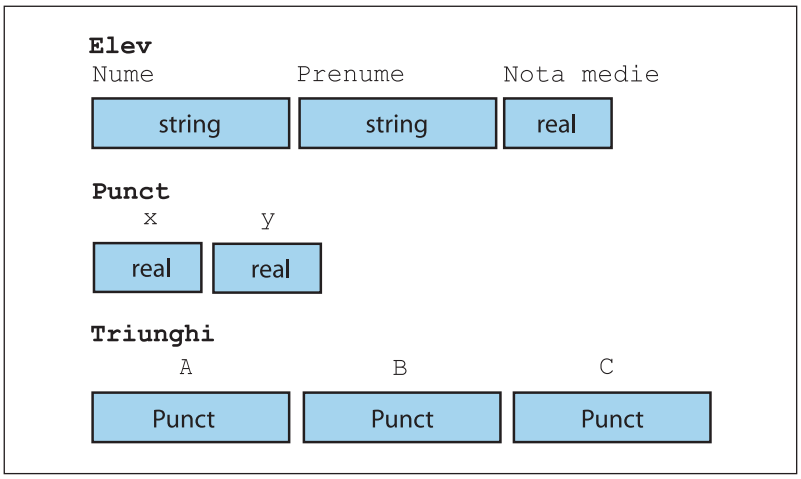


Fig. 1.4. Structura datelor de tip `Elev`, `Punct` şi `Triunghi`

Fiind date două variabile de tip *articol* de acelaşi tip, numele variabilelor pot apărea într-o instrucțiune de atribuire. Această atribuire înseamnă copierea tuturor câmpurilor din membrul drept în membrul stâng. De exemplu, pentru tipurile de date şi variabilele declarate mai sus instrucțiunile

```
E1:=E2;
T2:=T3;
P2:=P1
```

sunt corecte.

Fiecare componentă a unei variabile de tip **`record`** poate fi specificată explicit, prin numele variabilei şi denumirile de câmpuri, separate prin puncte.

Exemple:

1) `E1.Nume, E1.Prenume, E1.NotaMedie;`

2) `E2.Nume, E2.Prenume, E2.NotaMedie;`

3) `P1.x, P1.y, P2.x, P2.y;`

4) `T1.A, T1.B, T1.C, T2.A, T2.B, T2.C;`

5) `T1.A.x, T1.A.y, T2.B.x, T2.B.y.`

Evident, componenta `E1.Nume` este de tip **`string`**; componenta `P1.x` este de tip `real`; componenta `T1.A` este de tip `Punct`; componenta `T1.A.x` este de tip `real` ş.a.m.d.

Asupra componentelor datelor de tip *articol* se pot efectua toate operaţiile admise de tipul câmpului respectiv. Programul ce urmează compară notele medii a doi elevi şi afişează pe ecran numele şi prenumele elevului cu nota medie mai bună. Se consideră că elevii au note medii diferite.

```
Program P85;
{ Date de tipul Elev }
type Elev = record
              Nume : string;
              Prenume : string;
              NotaMedie : real;
            end;
var E1, E2, E3 : Elev;
begin
  writeln('Dati datele primului elev:');
  write('Numele:');     readln(E1.Nume);
  write('Prenumele:');  readln(E1.Prenume);
  write('Nota medie:'); readln(E1.NotaMedie);

  writeln('Dati datele elevului al doilea:');
  write('Numele:');     readln(E2.Nume);
  write('Prenumele:');  readln(E2.Prenume);
  write('Nota medie:'); readln(E2.NotaMedie);

  if E1.NotaMedie > E2.NotaMedie then E3:=E1 else E3:=E2;

  writeln('Elevul cu media mai buna:');
  writeln(E3.Nume, ' ', E3.Prenume, ':', E3.NotaMedie : 5:2);
  readln;
end.
```

Orice tip de date **`record`** poate servi ca tip de bază pentru formarea altor tipuri structurate.

Exemplu:

```
type ListaElevilor = array [1..40] of Elev;
var  LE : ListaElevilor;
```

Evident, notaţia `LE[i]` specifică elevul `i` din listă; notaţia `LE[i].Nume` specifică numele acestui elev, notaţia `LE[i].Prenume` specifică prenumele elevului respectiv ş.a.m.d. Programul ce urmează citeşte de la tastatură datele referitoare la *n* elevi şi afişează pe ecran numele, prenumele şi nota medie a celui mai bun elev. Se consideră că elevii au note medii diferite.

```
Program P86;
{ Tablou cu componente de tipul Elev }
type Elev = record
              Nume : string;
              Prenume : string;
              NotaMedie : real;
            end;
     ListaElev = array [1..40] of Elev;
var E : Elev;
    LE : ListaElev;
    n : 1..40;
    i : integer;
begin
  write('n='); readln(n);
  for i:=1 to n do
    begin
      writeln('Dati datele elevului  ', i);
      write('Numele:  ');       readln(LE[i].Nume);
      write('Prenumele:  ');    readln(LE[i].Prenume);
      write('Nota Medie:  ');   readln(LE[i].NotaMedie);
    end;
  E.NotaMedie:=0;
  for i:=1 to n do
    if LE[i].NotaMedie > E.NotaMedie then E:=LE[i];
  writeln('Cel mai bun elev:');
  writeln(E.Nume, ' ', E.Prenume, ':', E.NotaMedie : 5:2);
  readln;
end.
```

În general, un tip de date *articol* se defineşte cu ajutorul diagramelor sintactice din *figura 1.5*. În completare la articolele cu un număr fix de câmpuri, limbajul PASCAL permite utilizarea articolelor cu variante. Aceste tipuri de date se studiază în cursurile avansate de informatică.

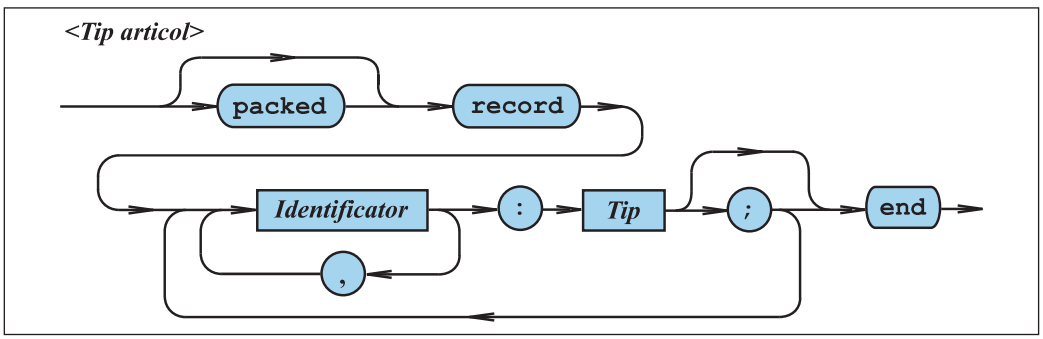


Fig. 1.5. Diagrama sintactică <*Tip articol*>

C++

În limbajul C++ un tip de date *articol* se defineşte printr-o structură gramaticală de forma:

struct [<*Nume tip*>] {
T_1 <*Nume câmp 1*> ;
T_2 <*Nume câmp 2*>;
...
T_n <*Nume câmp n*>;
}
[lista variabile];

unde T_1, T_2, .., T_n specifică tipul câmpurilor respective. Tipul unui nume de câmp este arbitrar, astfel un câmp poate să fie, la rândul lui, tot de tip *articol*. Prin urmare, se pot defini tipuri imbricate.

Exemple:

1)
```
struct Elev {
          string Nume[20];
          string Prenume[25];
          float NotaMedie;
         };
Elev E1, E2;
```

2)
```
struct Punct {
           double x;     // coordonata x
           double y;     // coordonata y
          };
Punct P1, P2;
```

3)
```
struct Triunghi {
                  Punct A;   // vârful A
                  Punct B;   // vârful B
                  Punct C;   // vârful C
                 };
Triunghi T1, T2, T3;
```

Structura datelor din exemplele în studiu este prezentată în *figura 1.4**.

Fiind date două variabile de tip *înregistrare* de acelaşi tip, numele variabilelor pot apărea într-o instrucţiune de atribuire. Această atribuire înseamnă copierea tuturor câmpurilor din membrul drept în membrul stâng. De exemplu, pentru tipurile de date şi variabilele declarate mai sus instrucţiunile

```
E1=E2;
T2=T3;
P2=P1
```

sunt corecte.

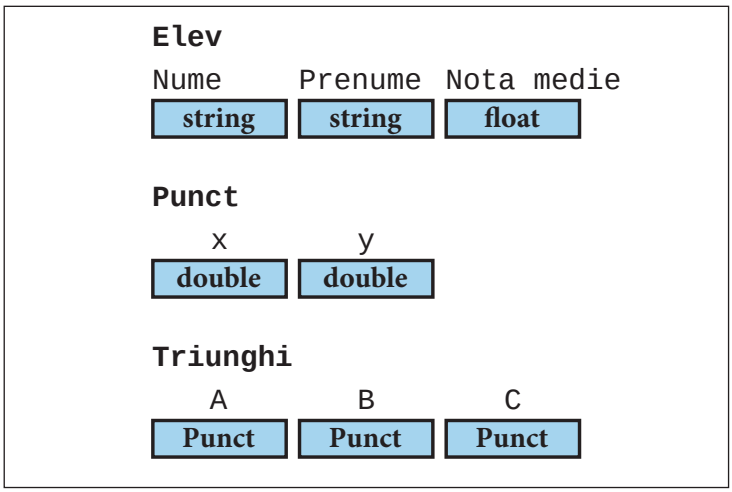


*Fig. 1.4**. Structura datelor de tip `Elev`, `Punct` și `Triunghi`

Fiecare componentă a unei variabile de tip **`struct`** poate fi specificată explicit, prin numele variabilei şi denumirile de câmpuri separate prin puncte.

Exemple:

1) `E1.Nume, E1.Prenume, E1.NotaMedie`

2) `E2.Nume, E2.Prenume, E2.NotaMedie`

3) `P1.x, P1.y, P2.x, P2.y`

4) `T1.A, T1.B, T1.C, T2.A, T2.B, T2.C`

5) `T1.A.x, T1.A.y, T2.B.x, T2.B.y`

6) `V[3].Nume, v[3].Prenume, V[3].NotaMedie`

Evident, componenta `E1.Nume` este de tip `string`; componenta `P1.x` este de tip **`double`**; componenta `T1.A` este de tip `Punct`; componenta `T1.A.x` este de tip **`double`** ş.a.m.d.

Asupra componentelor datelor de tip **`struct`** se pot efectua toate operațiile admise de tipul câmpului respectiv. Programul ce urmează compară notele medii a doi elevi şi afişează pe ecran numele şi prenumele elevului cu nota medie mai bună. Se consideră că elevii au note medii diferite.

```
//Program P85
// Date de tipul Elev
#include <iostream>
using namespace std;
int main()
{
```

```
  struct Elev {
              char Nume[20];
              char Prenume[30];
              float NotaMedie;
             };
  Elev E1, E2, E3;
  cout<<"Dati datele primului elev:"<<endl;
  cout<<"Numele:"<<endl;          cin>>E1.Nume;
  cout<<"Prenumele:"<<endl;       cin>>E1.Prenume;
  cout<<"Nota medie:"<<endl;      cin>>E1.NotaMedie;
  cout<<endl<<"Dati datele elevului al doilea:"<<endl;
  cout<<"Numele:"<<endl;          cin>>E2.Nume;
  cout<<"Prenumele:"<<endl;       cin>>E2.Prenume;
  cout<<"Nota medie:"<<endl;      cin>>E2.NotaMedie;
  if (E1.NotaMedie > E2.NotaMedie) E3=E1; else E3=E2;
  cout<<"Elevul cu media mai bună:"<<endl;
  cout<<E3.Nume<<' '<<E3.Prenume<<' '<<E3.NotaMedie;
  return 0;
}
```

Orice tip de date **`struct`** poate servi ca tip de bază pentru formarea altor tipuri structurate.

Exemplu:

```
typedef Elev ListaElevilor[40];
ListaElevilor LE;
```

Evident, notația `LE[i]` specifică elevul `i` din listă; notația `LE[i]`.Nume specifică numele acestui elev, notația `LE[i].Prenume` specifică prenumele elevului respectiv ş.a.m.d. Programul ce urmează citeşte de la tastatură datele referitoare la *n* elevi şi afişează pe ecran numele, prenumele şi nota medie a celui mai bun elev. Se consideră că elevii au note medii diferite.

```
//Program P86
// Tablou cu componente de tipul Elev
#include <iostream>
using namespace std;
int main()
{
struct Elev {
              char Nume[20];
              char Prenume[30];
              float NotaMedie;
             };
Elev E;
int i;
typedef Elev ListaElevilor[40];
```

```
ListaElevilor LE;
E.NotaMedie=0;
int n;
cout<<"n=";  cin>>n;
for (i=0; i<n; i++)
  {
   cout<<"Numele:";  cin>>LE[i].Nume;
   cout<<"Prenumele:";     cin>>LE[i].Prenume;
   cout<<"Nota medie:";    cin>>LE[i].NotaMedie;
  };
for (i=0; i<n; i++)
   if (E.NotaMedie < LE[i].NotaMedie) E = LE[i];
  cout << "Elevul cu nota medie mai mare:" << endl;
  cout << E.Nume << ' ' << E.Prenume << '  ' << E.NotaMedie
  << endl;
return 0;
}
```

În general, un tip de date *înregistrare* se definește cu ajutorul diagramelor sintactice din *figura 1.5**. În completare la articolele cu un număr fix de câmpuri, limbajul C++ permite utilizarea înregistrărilor cu structură variabilă, definite prin tipul de date **union**. Aceste tipuri de date se studiază în cursurile avansate de informatică.

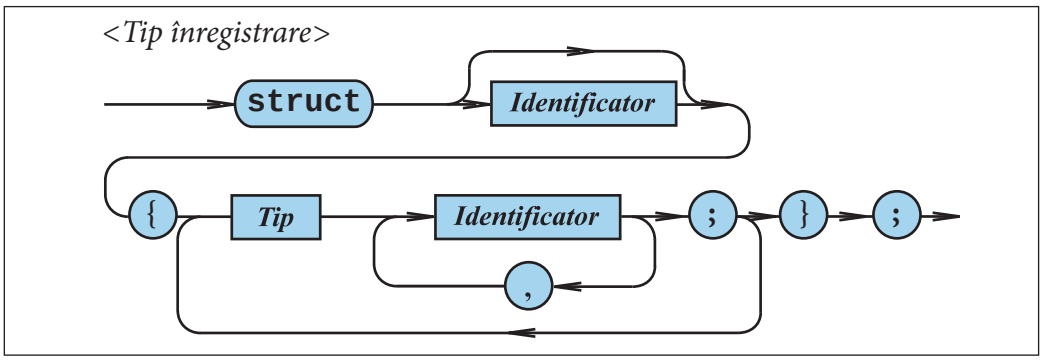


*Fig. 1.5**. Diagrama sintactică <*Tip articol*>

Întrebări şi exerciţii

❶ Care este mulțimea de valori ale unui tip de date *articol*?

❷ Scrieți formulele metalingvistice pentru diagramele sintactice din *figurile 1.5/1.5**.

❸ Indicați pe diagramele sintactice din *figurile 1.5/ 1.5** drumurile care corespund definițiilor tipurilor de date *articol* din programul P85.

❹ EXERSEAZĂ! Definiți tipul de date *articol* destinat reprezentării pe calculator a următoarelor informații:
a) coordonatele carteziene ale unui punct pe plan;
b) coordonatele carteziene ale unui punct în spațiu;

c) lista cărților din biblioteca personală: numele şi prenumele autorului, titlul cărții, limba, editura, anul ediției, țara în care a fost editată;
d) lista automobilelor dintr-un salon de vânzări auto: marca, tipul, culoarea caroseriei, capacitatea rezervorului, anul fabricației, prețul;
e) lista pieselor muzicale din mediateca personală: numele şi prenumele interpretului, titlul, genul de muzică, anul înregistrării;
f) lista pasagerilor unor curse aeriene: numele, prenumele, numărul cursei (şir de o lungime de până la opt caractere), data (ziua, luna, anul) şi ora decolării (ore, minute), tipul de avion, locul în avion (şir de o lungime de până la şase caractere);
g) lista angajaților unei companii: numele, prenumele, postul ocupat, data angajării (ziua, luna, anul), vârsta, adresa domiciliului (localitatea, strada, casa, apartamentul).

❺ APLICĂ! Se consideră următoarele declarații:

PASCAL

```
type Elev = record
        Nume, Prenume : string;
        T, P:real;
            end;
var E: Elev;
```

C++

```
struct elev {
             char Nume[20];
             char Prenume[25];
             float T, P;
            };
Elev E;
```

În aceste declarații câmpul `T` este nota medie în semestrul de toamnă, iar câmpul `P` – nota medie a elevului în semestrul de primăvară. Scrieți instrucțiunea de atribuire ce calculează nota medie anuală `M` a elevului E.

❻ REZOLVĂ! Se consideră următoarele tipuri de date

PASCAL

```
type
Data = record
          Ziua : 1..31;
          Luna : 1..12;
          Anul : integer;
       end;
Persoana = record
          NumePrenume : string;
          DataNasterii : Data;
           end;
ListaPersoane = array [1..50] of
Persoana;
```

C++

```
struct Data
  {
    int Ziua;Luna;Anul;
  };
struct Persoana {
    string NumePrenume;
    Data DataNasterii;
  };
Persoana ListaPersoane[50];
```

Elaborați un program care citeşte de pe tastatură datele referitoare la n persoane ($n \leq 50$) şi afişează pe ecran:
a) persoanele născute în ziua z a lunii;
b) persoanele născute în luna l a anului;
c) persoanele născute în anul a;
d) persoanele născute pe data $z.l.a$;
e) persoana cea mai în vârstă;
f) persoana cea mai tânără;

g) vârsta fiecărei persoane în ani, luni, zile;
h) lista persoanelor care au mai mult de *v* ani;
i) lista persoanelor în ordine alfabetică;
j) lista persoanelor ordonată conform datei naşterii;
k) lista persoanelor de aceeaşi vârstă (născuți în acelaşi an).

❼ Se consideră n puncte ($n \leq 30$) pe un plan euclidian. Fiecare punct i este definit prin coordonatele sale. Distanța dintre punctele i şi j se calculează după formula

$$d_{ij} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}.$$

Elaborați un program care afişează pe ecran punctele distanța dintre care este maximă.

❽ Aria triunghiului este dată de formula lui Heron

$$S = \sqrt{p(p-a)(p-b)(p-c)},$$

unde p este semiperimetrul, iar a, b şi c sunt lungimile laturilor respective. Utilizând tipurile de date `Punct` şi `Triunghi` din paragraful în studiu, elaborați un program care citeşte de la tastatură informațiile referitoare la n triunghiuri ($n \leq 10$) şi afişează pe ecran:
a) aria fiecărui triunghi;
b) coordonatele vârfurilor triunghiului cu aria maximă;
c) coordonatele vârfurilor triunghiului cu aria minimă;
d) informațiile referitoare la fiecare triunghi în ordinea creşterii ariilor.

1.5. Instrucţiunea `with`

NOTĂ
În limbajul C++ o astfel de instrucţiune nu există. Prin urmare, acest paragraf este destinat elevilor ce studiază doar limbajul PASCAL.

În limbajul PASCAL, componentele unei variabile de tip *articol* se specifică explicit prin numele variabilei şi denumirile de câmpuri separate prin puncte.

De exemplu, în prezența declarațiilor

```
type Angajat = record
                 NumePrenume : string;
                 ZileLucrate : 1..31;
                 PlataPeZi   : real;
                 PlataPeLuna : real;
               end;
var A : Angajat;
```

componentele variabilei `A` se specifică prin `A.NumePrenume`, `A.ZileLucrate`, `A.PlataPeZi` şi `A.PlataPeLuna`.

Întrucât numele `A` al variabilei de tip *articol* se repetă de mai multe ori, acest mod de referire a componentelor este, în anumite situații, incomod. Repetările obositoare pot fi evitate cu ajutorul instrucțiunii **`with`** (cu).

Sintaxa instrucțiunii în studiu este:

<*Instrucțiune* **with**> ::= **with** <*Variabilă*> {, <*Variabilă*>} **do** <*Instrucțiune*>

Diagrama sintactică este prezentată în *figura 1.6*.

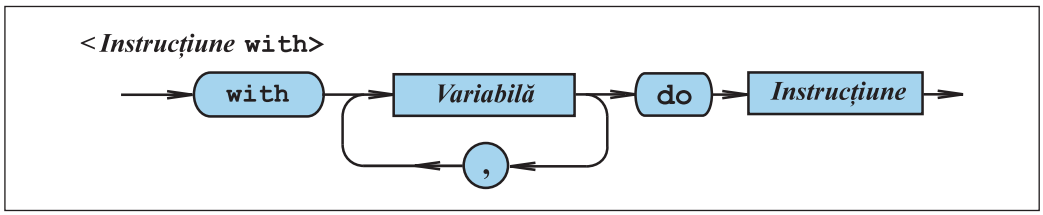


Fig. 1.6. Diagrama sintactică a instrucţiunii **with**

În interiorul unei instrucțiuni **with** componentele uneia sau a mai multe variabile de tip *articol* pot fi referite folosind numai numele câmpurilor respective.

Exemplu:

```
with A do PlataPeLuna:=PlataPeZi*ZileLucrate
```

Această instrucțiune este echivalentă cu următoarea:

```
A.PlataPeLuna:=A.PlataPeZi*A.ZileLucrate
```

Utilizarea instrucțiunii **with** necesită o atenție sporită din partea programatorului, care este obligat să specifice univoc componentele variabilelor de tip *articol*. În interiorul unei instrucțiuni **with**, la întâlnirea unui identificator, prima dată se testează dacă el poate fi interpretat ca un nume de câmp al articolului respectiv. Dacă da, identificatorul va fi interpretat ca atare, chiar dacă în acel moment este accesibilă şi o variabilă având acelaşi nume.

Exemplu:

```
type Punct = record
               x : real;
               y : real;
             end;
     Segment = record
                 A : Punct;
                 B : Punct;
               end;
var P : Punct;
    S : Segment;
    x : integer;
```

În cazul nostru identificatorul `x` poate să reprezinte fie variabila `x` de tip `integer`, fie câmpul `P.x` al articolului `P`.

În instrucțiunea

```
x:=1
```

identificatorul `x` se referă la variabila `x` de tip `integer`.

În instrucțiunea

```
with P do x:=1
```

identificatorul x se referă la câmpul P.x al variabilei de tip articol P.

Întrucât variabila de tip articol S nu conține niciun câmp cu numele S.x, în instrucțiunea

```
with S do x:=1
```

identificatorul x va fi interpretat ca variabila x de tip integer.

O instrucțiune de forma

with $v_1, v_2, ..., v_n$ **do** <*Instrucțiune*>,

unde $v_1, v_2, ..., v_n$ sunt variabile de tip *articol*, este echivalentă cu instrucțiunea

with v_1 **do**
with v_2 **do**
{...}
with v_n **do** <*Instrucțiune*>.

Evident, componentele variabilelor $v_1, v_2, ..., v_n$ trebuie specificate univoc prin denumirile câmpurilor respective.

De exemplu, pentru variabilele P şi S, declarate mai sus, putem scrie:

```
with P, S do
begin
  x:=1.0;    { referire la P.x }
  y:=1.0;
  A.x:=0;    { referire la S.A.x }
  A.y:=0;
  B.x:=2.0; { referire la S.B.x }
  B.y:=2.0;
end;
```

În mod obişnuit, instrucțiunea **with** se utilizează numai în cazurile în care se ajunge la o reducere semnificativă a textului unui program.

Întrebări şi exerciţii

❶ Indicați pe diagrama sintactică din *figura 1.6* drumurile care corespund instrucțiunilor **with** din exemplele paragrafului în studiu.

❷ Care este destinația instrucțiunii **with**?

❸ APLICĂ! Utilizând instrucțiunea **with**, excludeți din programele P85 şi P86 din paragraful precedent repetările de genul

```
E1.Nume, E1.Prenume, ...,
LE[i].Nume, LE[i].Prenume.
```

❹ Se consideră următoarele tipuri de date:

```
type Angajat = record
                 NumePrenume : string;
                 ZileLucrate : 1..31;
                 PlataPeZi : real;
                 PlataPeLuna : real;
               end;
     ListaDePlata = array [1..50] of Angajat;
```

Plata pe lună a fiecărui angajat se calculează înmulțind plata pe zi cu numărul de zile lucrate. Elaborați un program care:
a) calculează plata pe lună a fiecărui angajat;
b) calculează salariul mediu al angajaților incluşi în listă;
c) afişează pe ecran datele despre angajații cu plata lunară maximă;
d) afişează lista angajaților ordonată alfabetic;
e) afişează lista angajaților în ordinea creşterii plăților pe zi;
f) ordonează lista angajaților în ordinea creşterii plăților pe lună;
g) afişează lista angajaților în ordinea creşterii numărului de zile lucrate.

❺ REZOLVĂ! Un cerc poate fi definit prin coordonatele x, y şi raza r. Elaborați un program care citeşte de la tastatură datele referitoare la n cercuri ($n \leq 50$) şi afişează pe ecran:
a) coordonatele centrului şi raza cercului cu aria maximă;
b) numărul de cercuri incluse în cercul cu raza maximă şi coordonatele centrelor respective;
c) coordonatele centrului şi raza cercului cu aria minimă;
d) numărul de cercuri în care este inclus cercul cu raza minimă şi coordonatele centrelor respective.

1.6. Tipuri de date *mulțime*

Amintim că o mulțime este formată din obiecte fizice sau virtuale (ale gândirii) care au o proprietate comună. Obiectele din care este formată o mulțime se numesc *elementele mulțimii*. Elementele oricărei mulțimi sunt distincte, iar ordinea lor nu este importantă.

PASCAL

În limbajul PASCAL, un tip de date *mulțime* se defineşte în raport cu un tip de bază care trebuie să fie ordinal:

<*Tip mulțime*> ::= **[packed] set of** <*Tip*>

Valorile unui tip de date **set** sunt mulțimi formate din valorile tipului de bază. Dacă tipul de bază are n valori, tipul *mulțime* va avea 2^n valori. În implementările limbajului valoarea lui n este limitată, de regulă $n \leq 256$.

În PASCAL o mulțime poate fi specificată enumerându-i-se elementele între parantezele pătrate „[” şi „]”, care țin locul acoladelor din matematică.

Notația **[ ]** reprezintă mulțimea vidă.

Exemple:

```
type Indice = 1..10;
     Zi = (L, Ma, Mi, J, V, S, D);
     MultimeIndicii = set of Indice;
     ZileDePrezenta = set of Zi;
var MI : MultimeIndicii;
    ZP : ZileDePrezenta;
```

Tipul ordinal `Indice` are $n = 10$ valori: `1, 2, 3, 4, 5, 6, 7, 8, 9, 10`. Tipul `MultimeIndicii` are $2^{10} = 1\ 024$ de valori, şi anume:

```
[], [1], [2], ..., [1, 2], [1, 3], ...,
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10].
```

Prin urmare, variabila `MI` poate să aibă oricare dintre aceste valori, de exemplu:

```
MI:= [1, 3].
```

Tipul ordinal `Zi` are $n = 7$ valori: `L, Ma, Mi, J, V, S, D`. Tipul `ZileDePrezenta` are $2^7 = 128$ de valori, şi anume:

```
[], [L], [Ma], [Mi], ..., [L, Ma], [L, Mi], ...,
[L, Ma, Mi, J, V, S, D].
```

Variabila `ZP` poate să aibă oricare dintre aceste valori, de exemplu,

```
ZP:= [L, Ma, Mi, V]
```

O valoare de tip *mulțime* poate fi specificată printr-un **constructor** (generator) de mulțime. Diagrama sintactică a unității gramaticale <*Constructor mulțime*> este prezentată în *figura 1.7*.

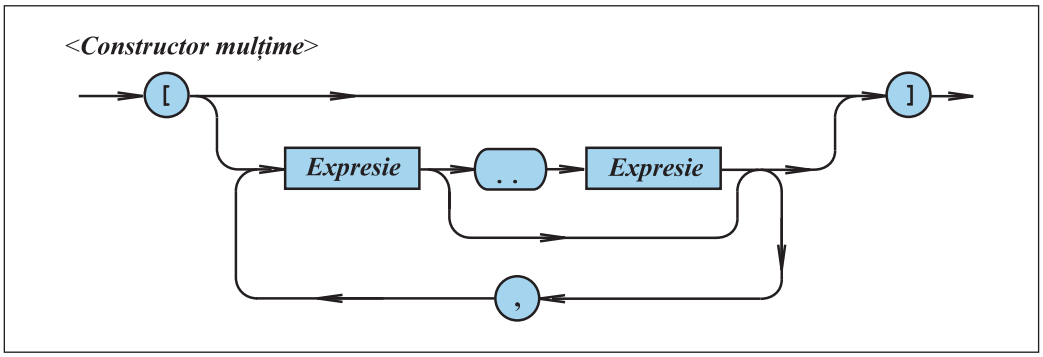


Fig. 1.7. Diagrama sintactică <*Constructor mulţime*>

Un constructor conține specificarea elementelor mulțimii, separate prin virgule şi incluse între paranteze pătrate. Un element poate să fie o valoare concretă a tipului de bază sau un interval de forma:

<*Expresie*>..<*Expresie*>.

Valorile expresiilor în studiu precizează limitele inferioare şi superioare ale intervalului.

Exemple :

1) `[];`

2) `[1, 2, 3, 8];`

3) `[1..4, 8..10];`

4) `[i-k..i+k];`

5) `[L, Ma, V..D].`

Asupra valorilor unui tip de date *mulțime* se pot efectua operațiile uzuale:
+ reuniunea;
* intersecția;
- diferența,
rezultatul fiind de tip *mulțime* şi operațiile relaționale:
= egalitatea;
<> inegalitatea;
<=, >= incluziunea;
in apartenența,
rezultatul fiind de tip `boolean`.

Programul ce urmează afişează pe ecran rezultatele operațiilor +, * şi -, efectuate asupra valorilor de tip `MultimeIndicii`.

```
Program P87;
{ Date de tip MultimeIndicii }
type Indice = 1..10;
     MultimeIndicii = set of Indice;
var A, B, C : MultimeIndicii;
    i : integer;
begin
  A:= [1..5, 8];      { A contine 1, 2, 3, 4, 5, 8 }
  B:= [1..3, 9, 10]; { B contine 1, 2, 3, 9, 10 }
  C:= [];             { C este o multime vida }
  C:=A+B;             { C contine 1, 2, 3, 4, 5, 8, 9, 10 }

  writeln('Reuniune');
  for i:=1 to 10 do
    if i in C then write(i:3);
  writeln;

  C:=A*B;             { C contine 1, 2, 3 }
  writeln('Intersectie');
  for i:=1 to 10 do
    if i in C then write(i:3);
  writeln;
```

```
  C:=A-B;              { C contine 4, 5, 8 }
  writeln('Diferenta');
  for i:=1 to 10 do
    if i in C then write(i:3);
  writeln;
  readln;
end.
```

Spre deosebire de tablouri şi articole, componentele cărora pot fi referite direct, respectiv prin indicii şi denumiri de câmpuri, elementele unei mulțimi nu pot fi referite. Se admite numai verificarea apartenenței elementului la o mulțime (operația relațională **in**). În pofida acestui fapt, utilizarea tipurilor de date *mulțime* măreşte viteza de execuție şi îmbunătățeşte lizibilitatea programelor PASCAL. De exemplu, instrucțiunea:

```
if (c='A') or (c='E') or (c='I') or (c='O') or (c='U') then ...
```

poate fi înlocuită cu o instrucțiune mai simplă:

```
if c in ['A','E','I','O','U'] then ...
```

ŞTIAI CĂ?

Eratostene a fost un matematician, geograf, poet, astronom și muzician grec, care a trăit între anii 276 și 194 î.Hr. A fost primul om care a calculat circumferința pământului, a calculat înclinația axei de rotație a pământului, a formulat conceptul de an bisect, a desenat una dintre primele harți ale lumii. Eratostene a inventat un algoritm foarte eficient de calcul al tuturor numerelor prime ce nu depășesc un număr dat.

Sursa: https://www.britannica.com

Un alt exemplu sugestiv este utilizarea tipurilor de date *mulțime* în calcularea numerelor prime mai mici decât un număr natural dat *n*. Pentru aceasta se foloseşte algoritmul *Ciurul (sita) lui Eratostene*:

1) în sită se depun numerele 2, 3, 4, ..., *n*;
2) din sită se extrage cel mai mic număr *i*;
3) numărul extras se include în mulțimea numerelor prime;
4) din sită se elimină toți multiplii *m* ai numărului *i*;
5) procesul se încheie când sita s-a golit.

```
Program P88;
{ Ciurul (sita) lui Eratostene }
const n = 50;
```

```
type MultimeDeNumere = set of 1..n;
var Sita, NumerePrime : MultimeDeNumere;
    i, m : integer;
begin
 {1} Sita:= [2..n];
     NumerePrime:=[];
     i:=2;
     repeat
 {2}    while not (i in Sita) do i:=succ(i);
 {3}    NumerePrime:=NumerePrime+[i];
        write(i:4);
        m:=i;
 {4}    while m<=n do
          begin Sita:=Sita-[m]; m:=m+i; end;
 {5} until Sita=[];
     writeln;
     readln;
end.
```

Corespondența dintre punctele algoritmului şi instrucțiunile care le exprimă este indicată în comentariile din partea stângă a liniilor de program.

C++

În limbajul C++ mulțimile pot fi reprezentate prin tablouri unidimensionale. De obicei, componentele respective sunt declarate ca aparținând unor tipuri ordinale de date, adică **int**, **bool**, **char** şi **enum**.

Exemple:

1) mulțimile A formate din numere întregi ce conțin până la 100 de elemente pot fi reprezentate prin tabloul `int A[100]`;

2) mulțimile B formate din cele 26 de caractere ale alfabetului latin pot fi reprezentate prin tabloul `char B[26]`;

3) Mulțimile Z formate din valorile tipului de date `enum Zi {L, Ma, Mi, J, V, S, D}` pot fi reprezentate prin tabloul `Zi Z[7]`.

Evident, în afară de tabloul propriu-zis, în care sunt memorate elementele mulțimii supuse prelucrării, va trebui să mai declarăm o variabilă întreagă n în care să memorăm numărul curent de elemente ale mulțimii respective.

În cazul mulțimilor A, B şi Z descrise mai sus, pot fi declarate variabile n_A, n_B şi n_Z, de exemplu: `int nA, nB, nZ`.

Prezentăm în continuare un program care citeşte de la tastatură mulțimile A şi B, formate din cel mult 20 de numere întregi, şi afişează pe ecran intersecția acestora, adică mulțimea $C = A \cap B$.

Amintim că mulțimea C trebuie să conțină doar elementele care apar atât în mulțimea A, cât şi în mulțimea B.

```
// Program P87
// Reprezentarea multimilor prin tablouri
// Intersectia multimilor A si B
```

```
#include <iostream>
using namespace std;
int main ()
{
    int A[20], B[20], C[20],
        nA, nB, nC,
        i, j, k;  // contoare cicluri
    bool Gasit;   // fanion: elementul multimii A
                  // se regaseste in mulţimea B
    cout << "Dati numarul de elemente ale multimii A: ";
    cin >> nA;
    cout << "Dati elementele multimii A:" << endl;
    for (i=0; i<nA; i++) cin >> A[i];
    cout << "Dati numarul de elemente ale multimii B: ";
    cin >>nB;
    cout << "Dati elementele multimii B:" << endl;
    for (j=0; j<nB; j++) cin >> B[j];
    nC=0; // initial multimea C este vida
    for (i=0; i<nA; i++)
      {
       Gasit = false;
       for (j=0; j<nB; j++) if (A[i]==B[j]) Gasit = true;
       if (Gasit==true) { C[nC] = A[i]; nC++; };
      }
    cout << "Numarul de elemente ale multimii C = " << nC << endl;
    cout << "Multimea C:" << endl;
    for (k=0; k<nC; k++) cout << C[k] << " ";
    cout << endl;
    return 0;
}
```

O altă metodă de reprezentare a mulţimilor constă în utilizarea aşa-numitor **vectori caracteristici**. În această metodă se presupune că există o mulţime universală (*universum*), notată prin U, ce conţine toate elementele din care pot fi formate mulţimile de prelucrat.

De exemplu, dacă se preconizează prelucrarea mulţimilor, elementele cărora sunt numere naturale mai mici ca 100, atunci $U = \{0, 1, 2, 3, \ldots, 99\}$. Într-un mod similar, dacă se presupune prelucrarea mulţimilor, elementele cărora sunt literele mari ale alfabetului latin, atunci $U = \{A, B, C, \ldots, Z\}$.

Presupunem că mulţimea universală conţine n elemente:

$$U = \{u_1, u_2, u_3, \ldots, u_n\},$$

iar mulţimea de prelucrat conţine m elemente:

$$A = \{a_1, a_2, a_3, \ldots, a_m\}.$$

Pe calculator, această mulţime poate fi reprezentată prin vectorul caracteristic

$$V = (v_1, v_2, v_3, \ldots v_i, \ldots, v_n),$$

unde

$$v_i = \begin{array}{l} 1, \text{ dacă } u_i \in A; \\ 0, \text{ în caz contrar.} \end{array}$$

Cu alte cuvinte, componenta i a vectorului caracteristic V are valoarea 1 dacă elementul respectiv din mulțimea universală aparține mulțimii A şi valoarea 0 în caz contrar.

De exemplu, dacă $U = \{1, 2, 3, 4, 5\}$ şi $A = \{2, 4\}$, atunci vectorul caracteristic:

$$V = (0, 1, 0, 1, 0).$$

Într-un mod similar, dacă $U = \{a, b, c, d, e, f\}$ şi $B = \{a, c, e, f\}$, atunci vectorul caracteristic:

$$V = (1, 0, 1, 0, 1, 1).$$

Din exemplele de mai sus se observă că numărul de componente ale vectorului caracteristic V al unei mulțimi A coincide cu numărul de elemente n ale mulțimii universale U. Evident, numărul de componente nenule ale vectorului caracteristic V este egal cu numărul de elemente m ale mulțimii A.

Un exemplu sugestiv de utilizare a vectorilor caracteristici pentru reprezentarea mulțimilor de prelucrat este calcularea numerelor prime mai mici decât un număr natural dat n. Pentru aceasta se foloseşte algoritmul *Ciurul (sita) lui Eratostene*:

1) în sită se depun numerele 2, 3, 4, ..., n;
2) din sită se extrage cel mai mic număr i;
3) numărul extras se include în mulțimea numerelor prime;
4) din sită se elimină toți multiplii m ai numărului i;
5) procesul se încheie când sita s-a golit.

```
// Program P88
// Ciurul (sita) lui Eratostene
// Reprezentarea multimilor prin vectori caracteristici
// Universum U = {0, 1, 2, ..., n}
#include <iostream>
using namespace std;
int main()
{
  const int nmax = 51;
  int n, S[nmax], // Vectorul caracteristic Sita
      P[nmax],    // Vectorul caracteristic Numere prime
      i,          // Numarul extras din Sita
      j, k;       // Contoare ciclu
  cout << "Dati n= "; cin >> n;
  // Depunem numerele 2, 3, 4, .., n in Sita
  S[0] = 0; S[1] = 0;
  for (j = 2; j <= n; j++) S[j] = 1;
  // Zerografiem multimea numerelor prime
  for (j = 0; j <= n; j++) P[j] = 0;
  // Extragem din sita cel mai mic numar i
```

```
  for (i = 2; i <= n; i++)
    {
      if (S[i] != 0)
        {
          P[i] = 1; // includem i in Numere prime
          S[i] = 0; // excludem i din Sita
          // Eliminam din Sita toti multiplii lui i
          for (j = i; j <= n; j++) if ((j % i)==0) S[j] = 0;
        }
    }
  cout << "Numerele prime:" << endl;
  for (j = 2; j <= n; j++) if (P[j] == 1) cout << j << ' ';
  cout << endl;
  return 0;
}
```

Întrebări şi exerciţii

❶ OBSERVĂ! (PASCAL) Enumerați valorile posibile ale variabilelor din declarațiile ce urmează:

```
var V : set of 'A'..'C';
    S : set of (A, B, C);
    I : set of '1'..'2';
    J : set of 1..2;
```

❷ OBSERVĂ! (PASCAL) Comentați următorul program:

```
Program P89;
{ Eroare }
type Multime = set of integer;
var M : Multime;
    i : integer;
begin
  M:=[1, 8, 13];
  for i:=1 to MaxInt do
    if i in M then writeln(i);
end.
```

❸ EXERSEAZĂ! (PASCAL) Se consideră următoarele declarații:

```
type Culoare = (Galben, Verde, Albastru, Violet);
     Nuanta = set of Culoare;
var NT : Nuanta;
```

Listați valorile posibile ale variabilei `NT`.

❹ EXERSEAZĂ! (PASCAL) Scrieți formula metalingvistică care corespunde diagramei sintactice <*Constructor mulțime*> din *figura 1.7*.

❺ EXERSEAZĂ! (PASCAL) Se consideră tipul de date `MultimeIndicii` din paragraful în studiu. Precizați mulțimile specificate de constructorii ce urmează:

a) `[];`
b) `[1..10];`
c) `[1..3, 9..10];`
d) `[1+1, 4..7, 9];`
e) `[3, 7..9];`
f) `[4..3];`
g) `[1..3, 7..6, 9];`
h) `[4-2..7+1];`
i) `[7-5..4+4];`
j) `[6, 9, 1..2].`

❻ EXERSEAZĂ! (C++) Se consideră mulțimea universală U = {L, Ma, Mi, J, V, S, D}, care reprezintă zilele săptămânii. Scrieți vectorii caracteristici ai următoarelor mulțimi:
A = {S, D} – zilele de odihnă ale săptămânii;
B = {L, Ma, Mi, J, V} – zilele de lucru ale săptămânii;
C = {L, Mi, J} – zilele în care au loc consultații online la disciplina şcolară Informatica;
D = {Ma, Mi} – zilele în care îşi desfăşoară activitatea cercurile artistice ale liceului;
E = {Ma, J, V} – zilele în care îşi desfăşoară activitatea secțiile sportive ale liceului;
F = {L, V} – zilele în care biblioteca liceului organizează expoziții de carte.

❼ EXERSEAZĂ! (C++) Utilizând mulțimile U, A, B, C, D, E, F din exercițiul precedent, scrieți vectorii caracteristici ai mulțimilor ce conțin:
G – zilele săptămânii în care îşi desfăşoară activitatea atât cercurile artistice, cât şi secțiile sportive ale liceului;
H – zilele săptămânii în care îşi desfăşoară activitatea fie cercurile artistice, fie secțiile sportive ale liceului;
I – zilele săptămânii în care atât cercurile artistice, cât şi secțiile sportive nu activează.
Elaborați un program care calculează şi afişează aceste mulțimi pe ecran.

❽ PROGRAMEAZĂ! Scrieți un program care citeşte de la tastatură elementele mulțimilor A şi B şi afişează pe ecran:
a) intersecția mulțimilor A şi B;
b) reuniunea mulțimilor A şi B;
c) diferența mulțimilor A şi B.
Mulțimile A şi B sunt formate din numere întregi.

❾ PROGRAMEAZĂ! Scrieți un program care citeşte de la tastatură elementele mulțimilor A şi B şi afişează pe ecran:
a) intersecția mulțimilor A şi B;
b) reuniunea mulțimilor A şi B;
c) diferența mulțimilor A şi B;
Mulțimile A şi B sunt formate din literele mari ale alfabetului latin.

❿ Elaborați un program care afişează pe ecran toate submulțimile mulțimii {1, 2, 3, 4}.

⓫ Elaborați un program care afişează pe ecran toate submulțimile mulțimii {*'A'*, *'B'*, *'C'*, *'D'*}.

⓬ Se consideră şiruri de caractere formate din literele mari şi mici ale alfabetului latin, în care cuvintele sunt separate prin spațiu, punct, virgulă, punct şi virgulă,

semnul exclamării sau semnul întrebării. Elaborați un program care afișează pe ecran cuvintele din șirurile citite de la tastatură.

⓭ Se consideră șiruri de caractere formate din literele mari și mici ale alfabetului latin. Elaborați un program care afișează pe ecran numărul de vocale din șirurile citite de la tastatură.

⓮ Elaborați un program care citește de la tastatură două șiruri de caractere și afișează pe ecran:
a) caracterele care se întâlnesc cel puțin în unul dintre șiruri;
b) caracterele care apar în ambele șiruri;
c) caracterele care apar în primul și nu apar în șirul al doilea.

⓯ Se consideră că numele unei persoane este scris corect dacă el este format din literele mari și cele mici ale alfabetului latin, începe cu literă mare, iar literele ce urmează sunt mici. Scrieți un program care verifică dacă șirul de caractere citit de la tastatură reprezintă un nume corect de persoană. În cazul unui nume corect, la ecran se va afișa mesajul `CORECT`, iar în caz contrar – `INCORECT`.

⓰ În implementările actuale ale limbajului numărul valorilor tipului de bază al unui tip *mulțime* este limitat, obișnuit $n \leq 256$. În consecință, programul `P88` nu poate calcula numere prime mai mari decât n. Elaborați un program pentru calcularea numerelor prime din intervalul 8, ..., 10 000.
Indicație: Sita din algoritmul lui Eratostene poate fi reprezentată printr-un tablou, componentele căruia sunt mulțimi.

⓱ STUDIU DE CAZ. (C++) Scrieți un program care calculează numerele prime cu ajutorul algoritmului *Ciurul (sita) lui Eratostene* reprezentând mulțimile *Sita* și *Numere prime* nu prin vectori caracteristici, ci prin tablouri ce conțin elementele propriu-zise ale acestora. Comparați complexitatea programului scris cu programul care utilizează vectori caracteristici.

⓲ CREEAZĂ! Se consideră posturile de televiziune *A* și *B*. În anumite zile ale săptămânii, nu neapărat aceleași, fiecare dintre aceste posturi nu difuzează filme artistice. De exemplu, Postul *A* nu difuzează filme artistice în zilele de *luni* și *vineri*, iar postul *B* – în zilele de *marți* și *duminică*.
Scrieți un program care, pentru fiecare din posturile *A* și *B*, citește de la tastatură zilele săptămânii în care nu sunt difuzate filme artistice și afișează pe ecran zilele în care spectatorul are posibilitatea să vizioneze un film artistic pe cel puțin unul din posturi. Zilele săptămânii vor fi citite de la tastatură și afișate pe ecran în forma uzuală, adică *Luni*, *Marți*, *Miercuri* ș.a.m.d., fără a folosi numerele lor de ordine sau abrevieri.
Indicație: Utilizați mulțimi elementele cărora sunt de tipul ordinal de date *enumerare*.

⓳ DESCOPERĂ! (C++) Este cunoscut faptul că mulțimile pot fi reprezentate prin tablouri unidimensionale, componentele cărora conțin elementele propriu-zise ale acestor mulțimi. Totodată, cu fiecare tablou trebuie să fie asociată o mărime întreagă ce conține numărul curent de elemente ale mulțimii respective. De exemplu, în cazul mulțimii *A*, formată din cel mult 20 de numere întregi, cu tabloul `int A[20]` se asociază variabila `nA`.

Pentru a face programele mai intuitive, mulțimile pot fi reprezentate și cu ajutorul tipului de date *articol*, care are forma `struct Multime {int E[nmax], nE;}`. În acest tip de date tabloul `E` este destinat stocării elementelor, iar variabila `nE` – stocării numărului curent de elemente ale mulțimii respective.
Rescrieți programele C++ din acest paragraf utilizând pentru reprezentarea mulțimilor date de tipul *articol*.

⓴ ÎNVAȚĂ SĂ ÎNVEȚI! (C++) Biblioteca standard a limbajului C++ conține containerul special `Set`, destinat reprezentării şi prelucrării datelor de tip *mulțime*. Recomandăm elevilor pasionați de programare să studieze în mod individual sau în echipă mijloacele oferite de acest container. Încercați să transcrieți programele din acest paragraf cu ajutorul funcțiilor definite în acest container.

1.7. Generalităţi despre fişiere

Până în prezent, programele elaborate de noi citeau datele de prelucrat de la tastatură şi afişau rezultatele prelucrărilor pe ecran. Evident, datele introduse de la tastatură se memorau în variabilele din programele respective. Însă, odată cu terminarea execuției oricărui program, valorile variabilelor din acest program se pierd, fapt care cere introducerea repetată a datelor de intrare la o nouă lansare în execuție a acestuia. În cazul unor date de intrare voluminoase, introducerile repetate de date cer un volum foarte mare de muncă.

Totodată, din experiența acumulată de noi în procesul prelucrării textelor, imaginilor, secvențelor audio şi video, cunoaştem deja că datele de prelucrat şi, desigur, rezultatele prelucrărilor, trebuie păstrate în fişiere. Amintim, fişierul reprezentă o colecție de date ce are un nume şi care se păstrează pe un suport extern de informație, de exemplu, pe discurile magnetice sau discurile optice, pe memoriile de tip *flash* ş.a.

Pentru a putea prelucra informațiile din fişierele de pe suporturile externe de informații, limbajele moderne de programare conțin tipuri structurate de date cu denumirea generică de *fişier*. În limbajele PASCAL şi C++, prin **fişier** se înțelege o structură de date care constă dintr-o secvență de componente (înregistrări). Numărul componentelor din secvență nu este fixat, însă sfârşitul secvenței este indicat de un simbol special, notat *EOF* (*End of File* – sfârşit de fişier). Fişierul care nu conține nicio componentă se numeşte *fişier vid*.

În ansamblu, utilizarea fişierelor are următoarele avantaje:

– se asigură păstrarea de lungă durată atât a datelor de intrare, cât şi a celor de ieşire, utilizatorul având posibilitatea să revină oricând la ele după terminarea procesului de execuție a programelor respective;

– programele devin universale, întrucât unul şi acelaşi program poate fi utilizat fără modificări pentru prelucrarea datelor din diverse fişiere externe;

– prelucrările complexe pot fi efectuate prin divizarea sarcinilor între diferite programe mai simple, întrucât datele de ieşire ale unui program pot fi utilizate ca date de intrare pentru un alt program.

ATENŢIE!
Mijloacele destinate prelucrării fişierelor depind în mare măsură de specificul sistemelor de operare și mediile de programare utilizate. În consecinţă, rezultatele furnizate de programele ce prelucrează fişiere ar putea să difere de la un calculator la altul.
Prin urmare, elevii sunt încurajaţi să studieze mijloacele destinate prelucrării fişierelor prin **metoda experimentării pe calculator**. Se recomandă ca fiecare din elevi să scrie și să lanseze în execuţie mici programe ce operează cu datele de pe suporturile externe de informaţii.

PASCAL

În limbajul PASCAL, un tip de date *fişier* se defineşte printr-o declaraţie de forma:

<*Tip fişier*> ::= [**packed**] **file of** <*Tip*>;

unde <*Tip*> este tipul de bază, adică tipul componentelor fişierelor respective. Tipul de bază este un tip arbitrar, exceptând tipul *fişier* (nu există „fişier de fişiere").

Exemple:

1)
```
type FisierNumere = file of integer;
var FN : FisierNumere;
    n : integer;
```

2)
```
type FisierCaractere = file of char;
var FC : FisierCaractere;
    c : char;
```

3)
```
type Elev = record
                Nume : string;
                Prenume : string;
                NotaMedie : real;
            end;
     FisierElevi = file of Elev;
var FE : FisierElevi;
    E : Elev;
```

Structura datelor în studiu este prezentată în *figura 1.8*. Subliniem faptul că elementul *EOF*, care indică sfârşitul secvenţei, nu este o componentă a fişierului.

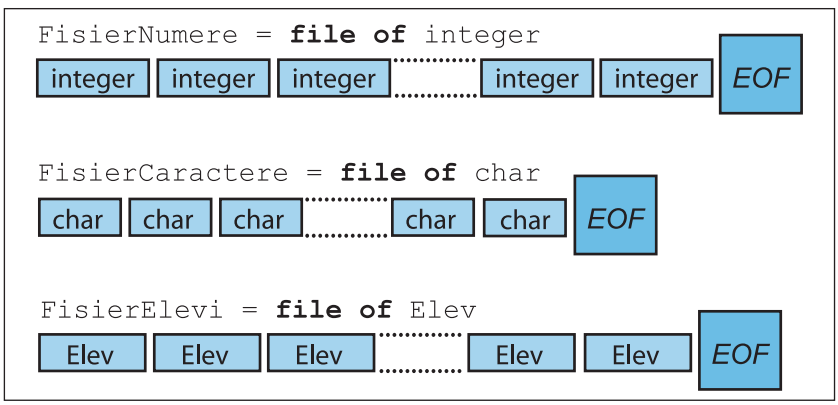


Fig. 1.8. Structura datelor de tip `FisierNumere`, `FisierCaractere` şi `FisierElevi`

Variabilele `FN`, `FC`, `FE` ş.a. de tip *fişier* se numesc **fişiere logice**, **fişiere PASCAL** sau, pur şi simplu, **fişiere**. Spre deosebire de celelalte tipuri de date, valorile cărora se păstrează în memoria internă a calculatorului, datele fişierelor PASCAL se păstrează pe suporturile de informaţie ale echipamentelor periferice (discuri şi benzi magnetice, discuri optice, hârtia imprimantei sau a dispozitivului de citit

documente ş.a.). Informaţia pe suporturile în studiu este organizată în formă de **fişiere externe** în conformitate cu cerinţele sistemului de operare. Prin urmare, înainte de a fi utilizată, o variabilă de tip fişier trebuie asociată cu un fişier extern. Metodele de asociere sunt specifice implementării limbajului şi sistemului de operare al calculatorului-gazdă.

În limbajul-standard asocierea se realizează prin includerea variabilelor de tip fişier ca argumente în antetul de program.

În Turbo PASCAL asocierea unei variabile de tip *fişier f* cu un fişier extern se realizează prin apelul de procedură

```
assign(f, s);
```

unde *s* este o expresie de tip **`string`** care specifică calea de acces şi numele fişierului extern.

Exemple:

1)
```
assign(FN, 'A:\REZULTAT\R.DAT')
```

– fişierul `FN` se asociază cu fişierul extern `R.DAT` din directorul `REZULTAT` de pe discul `A`.

2)
```
assign(FC, 'C:\A.CHR')
```

– fişierul `FC` se asociază cu fişierul `A.CHR` din directorul-rădăcină al discului `C`.

3)
```
write('Dati un nume de fisier:');
readln(str);
assign(FE, str);
```

– fişierul `FE` se asociază cu un fişier extern numele căruia este citit de la tastatură şi depus în variabila de tip **`string`** `str`.

După executarea instrucţiunii `assign(f, s)`, toate operaţiile referitoare la fişierul PASCAL *f* se vor efectua asupra fişierului extern *s*. Accentuăm faptul că dacă utilizatorul nu indică numele complet al fişierului extern (directorul în care se află şi numele propriu-zis al fişierului), el va fi creat/căutat în directorul implicit al mediului de dezvoltare a programelor PASCAL.

Cele mai uzuale operaţii asupra unui fişier sunt citirea şi scrierea unei componente.

Citirea unei componente se realizează printr-un apel de forma

```
read(f, v),
```

unde *v* este o variabilă declarată cu tipul de bază al fişierului *f*.

Scrierea unei componente se realizează printr-un apel de forma:

```
write(f, e),
```

unde *e* este o expresie asociată cu tipul de bază al fişierului *f*.

Exemple:

1)
```
read(FN, n);
```

2)
```
write(FC, c);
```

3)
```
read(FE, E).
```

După **tipul operațiilor permise** asupra componentelor, fişierele se clasifică în:
– fişiere de intrare (este permisă numai citirea);
– fişiere de ieşire (este permisă numai scrierea);
– fişiere de actualizare (sunt permise atât scrierea, cât şi citirea).

După **modul de acces la componente**, fişierele se clasifică în:
– fişiere cu acces secvențial sau secvențiale (accesul la componenta i este permis după ce s-a citit/scris componenta $i - 1$);
– fişiere cu acces aleatoriu sau direct (orice componentă se poate referi direct prin numărul ei de ordine i în fişier).

Menționăm că în limbajul-standard sunt permise numai fişiere secvențiale de intrare sau ieşire.

Tipul fişierului (de intrare, ieşire sau de actualizare) şi modul de acces (secvențial sau direct) se stabilesc printr-o operație de validare, numită **deschidere a fişierului**. Pentru aceasta, în limbajul-standard se utilizează următoarele proceduri:

`reset(` *f*`)` – pregăteşte un fişier existent pentru citire;

`rewrite(` *f*`)` – creează un fişier vid şi îl pregăteşte pentru scriere.

Când prelucrarea componentelor se termină, fişierul trebuie închis. La **închiderea** unui fişier sistemul de operare înscrie elementul *EOF*, înregistrează fişierul extern nou-creat în directorul respectiv ş.a.m.d.

În limbajul-standard se consideră că fişierele vor fi închise implicit la terminarea execuției programului respectiv. În Turbo PASCAL fişierul *f* se închide prin apelul de procedură `close(` *f*`)`.

În concluzie, prezentăm ordinea în care trebuie apelate procedurile destinate prelucrării datelor de tip *fişier*:

1. **Asocierea** fişierului PASCAL *f* cu fişierul extern *s*: `assign(` *f*, *s*`)`.

2. **Deschiderea** fişierului *f* pentru citire sau scriere: `reset(` *f* `)`, respectiv, `rewrite(` *f* `)`.

3. **Citirea/scrierea** unei componente a fişierului *f*: `read(` *f*, *v* `)`, respectiv, `write(` *f*, *e* `)`.

4. **Închiderea** fişierului *f*: `close(` *f* `)`.

După închiderea fişierului, variabila *f* poate fi asociată cu un alt fişier extern.

Întrucât valorile variabilelor de tip fişier se păstrează pe suporturile externe de informație, în PASCAL **atribuirile de fişiere sunt interzise**.

C++

În limbajul C++ un fişier este alcătuit din înregistrări (componente), care pot avea dimensiuni fixe sau variabile. Numărul înregistrărilor dintr-un fişier este limitat de capacitatea de memorare a suportului fizic folosit.

După **tipul operațiilor permise asupra componentelor**, fişierele se clasifică în:

– fişiere de intrare (este permisă numai citirea);
– fişiere de ieşire (este permisă numai scrierea);
– fişiere de actualizare (sunt permise atât scrierea, cât şi citirea).

După **modul de acces la componente**, fişierele se clasifică în:

– fişiere cu acces secvenţial (înregistrările pot fi prelucrate numai în ordinea în care ele sunt stocate în fişier);

– fişiere cu acces direct (componentele pot fi prelucrate în orice ordine). În acest caz, înainte de fiecare operaţie de citire/scriere se indică informaţia necesară pentru a selecta componenta ce urmează a fi prelucrată.

După **modul de interpretare a conţinuturilor**, fişierele se clasifică în:

– fişiere text care conţin numai caractere structurate pe linii;
– fişiere binare în care informaţia este văzută ca o colecţie de octeţi.

Asupra fişierelor pot fi efectuate următoarele **operaţii**:

– deschiderea fişierului;
– citirea datelor din fişier/scrierea datelor în fişier;
– închiderea fişierului.

În limbajul C++ prelucrarea datelor din fişierele externe se realizează cu ajutorul fluxurilor (*stream*-urilor).

Fluxurile sunt obiecte care transportă şi formatează şiruri de octeţi. Pentru exemplificare, amintim fluxurile pe care le cunoaştem deja:

`cin` – fluxul destinat citirii datelor de pe suportul extern de informaţie al dispozitivului standard de intrare, de obicei, tastatura;

`cout` – fluxul destinat scrierii datelor pe suportul extern de informaţie al dispozitivului standard de ieşire, de obicei, ecranul.

În general, programatorul poate crea şi fluxuri proprii. Astfel de fluxuri trebuie asociate cu fişierele de pe dispozitivele externe, de exemplu, cu fişierele de pe discurile magnetice sau din memoriile de tip *flash*. După asociere, operaţia de includere a datelor într-un flux are ca efect scrierea acestora în fişierul asociat, iar extragerea datelor dintr-un flux – citirea acestora din fişierul respectiv.

Amintim că includerea datelor într-un flux se efectuează cu ajutorul operatorului <<, iar extragerea datelor – cu ajutorul operatorului >>.

Pentru a avea acces la funcţiile destinate lucrului cu fluxuri şi fişiere, în programele C++ trebuie inclusă directiva:

```
#include <fstream>
```

Crearea unui flux de intrare şi asocierea lui cu un fişier extern se efectuează cu ajutorul următoarei construcţii gramaticale:

```
ifstream <Nume flux de intrare> ("<Nume fişier extern>");
```

Evident, fişierul extern din care vor fi citite datele trebuie să existe pe suportul de informaţii până la lansarea programului respectiv în execuţie. În caz contrar, în procesul executării programului, în momentul asocierii fluxului de intrare cu fişierul extern se va declanşa o eroare.

Crearea unui flux de ieşire şi asocierea lui cu un fişier extern se efectuează cu ajutorul următoarei construcţii gramaticale:

```
ofstream <Nume flux de ieşire> ("<Nume fişier extern>");
```

Dacă în momentul asocierii fluxului de ieşire cu fişierul extern ultimul încă nu există, el va fi creat în mod automat. Dacă fişierul extern există deja, toate înregistrările din el vor fi şterse.

Exemple:

1)
```
ifstream FI("C:\REZULTAT\R.DAT");
```

– crearea fluxului de intrare `FI` şi asocierea lui cu fişierul extern `R.DAT` din directorul `REZULTAT` de pe discul `C:`.

2)
```
ofstream FE("D:\D.CHR");
```

– crearea fluxului de ieşire `FE` şi asocierea lui cu fişierul extern `D.CHR` din directorul-rădăcină al discului `D:`.

3)
```
cout << ("Dati un nume de fisier:");
cin << (Str);
ofstream FE(Str);
```

– crearea fluxului de ieşire `FE`. Fluxul `FE` se asociază cu un fişier extern numele căruia este citit de la tastatură şi depus în variabila de tip `string Str`.

Accentuăm faptul că în ultimul exemplu numele fişierului extern nu este cunoscut la momentul scrierii programului, el fiind citit de la tastatură doar în momentul execuţiei acestuia. O astfel de abordare permite elaborarea de programe ce pot prelucra datele nu doar dintr-un anumit fişier extern, ci din oricare alt fişier indicat de utilizator.

Variabilele (fluxurile) `FI` şi `FE` din exemplele de mai sus se mai numesc **fişiere logice** sau, pur şi simplu, **fişiere**. Spre deosebire de celelalte tipuri de date, valorile cărora se păstrează în memoria internă a calculatorului, datele fişierelor se păstrează pe suporturile de informaţie ale echipamentelor periferice (discuri şi benzi magnetice, discuri optice, memorii *flash* ş.a.). Informaţia pe suporturile în studiu este organizată în formă de **fişiere externe** în conformitate cu cerinţele sistemului de operare.

Accentuăm faptul că dacă utilizatorul nu indică numele complet al fişierului extern (directorul în care se află şi numele propriu-zis al fişierului), el va fi creat/căutat în directorul implicit al mediului de dezvoltare a programelor C++.

Închiderea fişierelor asociate fluxurilor se efectuează prin apeluri de forma:

```
<Nume flux>.close();
```

Exemple:

```
FI.close(); FE.close();
```

După ce un fişier a fost închis, în el nu mai pot fi scrise şi din el nu mai pot fi citite niciun fel de date. Însă aceste operaţii devin posibile după o nouă deschidere a fişierului în cauză. Cu alte cuvinte, scrierea sau citirea datelor, după caz, este posibilă doar în perioada în care fişierul respectiv este deschis.

Evident, un fişier ce anterior a fost deschis, iar apoi a fost închis poate fi din nou deschis cu oricare dintre fluxurile din programul în curs de elaborare, nu

neapărat doar cu cel cu care a fost asociat mai înainte. Cu alte cuvinte, fluxurile (fişierele logice) pot fi folosite pentru deschiderea oricăror fişiere externe, iar fişierele externe pot fi deschise de oricare dintre fluxurile din program.

Citirea datelor dintr-un fişier extern se efectuează cu ajutorul operatorului de extragere a datelor din fluxul de intrare asociat cu fişierul respectiv:

```
<Nume flux> >> <Variabilă> { >> <Variabilă> };
```

Exemple:

1)
```
FI >> x;
```

2)
```
FI >> x >> y;
```

3)
```
FI >> x >> y >> z;
```

Scrierea datelor într-un fişier extern se efectuează cu ajutorul operatorului de includere a datelor în fluxul de ieşire asociat cu fişierul respectiv:

```
<Nume flux> << <Expresie> { << <Expresie> };
```

Exemple:

1)
```
FE << x;
```

2)
```
FE << "x=" << x;
```

3)
```
FE << x " " << y << " " << z;
```

În concluzie, prezentăm ordinea în care trebuie apelate funcţiile destinate prelucrării datelor din fişiere:

1) **Asocierea** fluxului *f* cu fişierul extern *s* şi deschiderea acestuia pentru citire/scriere: `ifstream` *f*(*s*), respectiv, `ofstream` *f*(*s*).

2) **Citirea/scrierea** unei componente a fişierului asociat cu fluxul *f*: *f* >> *v*, respectiv, *f* << *e*.

3) **Închiderea** fişierului asociat cu fluxul *f*: *f*`.close()`.

După închiderea fişierului, fluxul *f* poate fi asociat cu un alt fişier extern.

Întrebări şi exerciţii

❶ Explicaţi termenii *fişier logic, fişier extern.*

❷ Unde se păstrează datele unui fişier? Care este destinaţia asignării?

❸ Cum se clasifică fişierele în funcţie de operaţiile permise şi modul de acces?

❹ Ce operaţii pot fi efectuate asupra fişierelor? Dar asupra componentelor unui fişier?

❺ Pentru ce este necesară operaţia de asociere a unui fişier logic cu un fişier extern?

❻ Pentru ce sunt necesare operaţiile de deschidere şi închidere a fişierelor? Cum se efectuează aceste operaţii?

❼ Indicaţi ordinea în care trebuie efectuate operaţiile de prelucrare a datelor dintr-un fişier de intrare? Dar dintr-un fişier de ieşire?

PASCAL

❽ EXERSEAZĂ! Reprezentați pe un desen similar celui din *figura 1.8.* structura următoarelor tipuri de date:

a)
```
type Tabel = array [1..5, 1..10] of real;
     FisierTabele = file of Tabel;
```

b)
```
type Multime = set of 'A'..'C';
     FisierMultimi = file of Multime;
```

c)
```
type Punct = record x, y:real end;
     Segment = record A, B:Punct end;
     FisierSegmente = file of Segment;
```

❾ Explicați destinația procedurilor `read` și `write`. Ce tip trebuie să aibă variabila *v* într-un apel de forma `read`(*f, v*)? Ce tip trebuie să aibă expresia *e* într-un apel de forma `write`(*f, e*)?

❿ ANALIZEAZĂ! Se consideră declarațiile:

```
type Numere = file of integer;
var F, FIN, FOUT : Numere;
    i, j, k : integer;
    x, y, z : integer;
```

Explicați ce operații va efectua programul în procesul execuției următoarelor secvențe de instrucțiuni:

a)
```
assign(F, 'EXP.DAT');
rewrite(F);
i:=10; write(F, i);
j:=20; write(F, j);
k:=30; write(F, k);
close(F);
```

b)
```
assign(F, 'EXP.DAT');
reset(F);
read(F, x);
read(F, y);
read(F, z);
close(F);
```

c)
```
assign(FOUT, 'EXP.DAT');
rewrite(FOUT);
i:=40; j:=50; k:=60;
write(FOUT, i); write(FOUT, j); write(FOUT, k);
close(FOUT);
assign(FIN, 'EXP.DAT');
reset(FIN);
read(FIN, x); read(FIN, y); read(FIN, z);
close(FIN);
```

⓫ În condițiile exercițiului precedent, ce valori vor lua variabilele `x`, `y`, `z` dacă secvențele de instrucțiuni *a), b)* și *c)* vor fi executate consecutiv, una după alta?

⓬ Variabilele A şi B sunt introduse prin declarația

```
var A, B : file of integer;
```

Este oare corectă instrucțiunea?

```
A:=B
```

Argumentați răspunsul.

C++

⓭ Cum se declară fluxurile? Cum se asociază un flux cu un fişier extern?

⓮ ANALIZEAZĂ! Se consideră declarațiile:

```
int i, j, k,
int x, y, z;
```

Explicați ce operații va efectua programul în procesul execuției următoarelor secvențe de instrucțiuni:

a)
```
ofstream F("EXP.DAT");
i:=10; F << i << " ";
j:=20; F << j << " ";
k:=30; F << k;
F.close();
```

b)
```
ifstream F("EXP.DAT");
F >> x;
F >> y;
F >> z;
F.close();
```

c)
```
ofstream FOUT("EXP.DAT");
i:=40; j:=50; k:=60;
FOUT << i << " " << j << " " << k;
FOUT.close();
ifstream FIN("EXP.DAT");
FIN >> x >> y >> z;
FIN.close();
```

⓯ ANALIZEAZĂ! În condițiile exercițiului precedent, ce valori vor lua variabilele x, y, z dacă secvențele de instrucțiuni *a)*, *b)* şi *c)* vor fi executate consecutiv, una după alta?

⓰ ANALIZEAZĂ! Variabilele A şi B sunt introduse prin declarația

```
fstream A("text.in");
ofstream B("test.out");
```

Este oare corectă instrucțiunea?

```
B=A
```

Argumentați răspunsul.

⓱ DESCOPERĂ! În afară de fluxurile de intrare (`ifstream`) şi cele de ieşire (`ofstream`), limbajul C++ oferă utilizatorului şi fluxuri de tipul `fstream`. Crearea unui flux de acest tip şi asocierea lui cu un fişier extern se efectuează cu ajutorul următoarei construcţii gramaticale:

`fstream` <*Nume flux de intrare/ieşire*> (”<*Nume fişier extern*>”, <*Opţiune*>);

unde <*Opţiune*> specifică modul de deschidere a fluxului. Utilizând sistemul de asistenţă al mediului de dezvoltare a programelor şi surse din Internet, aflaţi destinaţia opţiunilor ce pot fi utilizate la asocierea fluxurilor de acest tip cu fişierele externe. Completaţi tabelul de mai jos:

Opţiune	Destinaţia
`ios::in`	
`ios::out`	
`ios::app`	
`ios::ate`	
`ios::trunc`	
`ios::binary`	

⓲ DESCOPERĂ! În limbajul C++ există mai multe funcţii destinate verificării stării curente a unui flux. Utilizând sistemul de asistenţă al mediului de dezvoltare a programelor şi surse din Internet, aflaţi ce rezultate returnează aceste funcţii şi completaţi tabelul de mai jos:

Funcţia	Destinaţia
`bad()`	
`fail()`	
`eof()`	
`good()`	
`clear()`	

1.8. Fişiere secvenţiale

PASCAL

Fie definiţiile PASCAL

```
type FT = file of T;
var f : FT; v : T;
```

prin care au fost introduse tipul *fişier* `FT` cu tipul de bază `T`, variabila de tip *fişier* *f* şi variabila *v* de tipul `T`.

Pentru a deschide un **fişier secvenţial de ieşire**, se apelează procedura `rewrite(f)`. În continuare în fişier se înscriu componentele respective. O componentă se înscrie printr-un apel de forma:

`write(`f`,` e`)`,

unde e este o expresie de tipul `T`. O instrucțiune de forma

`write(`f`,` e_1`,` e_2`, ..., ` e_n`)`

este echivalentă cu secvența de instrucțiuni

`write(`f`,` e_1`) ; write(`f`,` e_2`); ...; write(`f`,` e_n`)`.

După înscrierea ultimei componente fişierul trebuie închis.

Exemplu:

```
Program P90;
{ Crearea unui fisier cu componente de tipul Elev }
type Elev = record
              Nume : string;
              Prenume : string;
              NotaMedie : real;
            end;
     FisierElevi = file of Elev;
var FE : FisierElevi;
    E : Elev;
    str : string;
    i, n : integer;
begin
  write('Dati numele fisierului de creat: ');
  readln(str);

  assign(FE, str);     { asociaza FE cu numele din str }
  rewrite(FE);         { deschide FE pentru scriere }

  write('Dati numarul de elevi: '); readln(n);

  for i:=1 to n do
    begin
      writeln('Dati datele elevului ', i);

          { citeste cimpurile variabilei E de la tastatura }
          write('Numele: ');      readln(E.Nume);
          write('Prenumele: ');   readln(E.Prenume);
          write('Nota medie: '); readln(E.NotaMedie);

       { scrie valoarea variabilei E in fisierul FE }
        write(FE, E);
    end;
  close(FE);           { inchide fisierul FE }
  readln;
end.
```

Pentru a **deschide un fişier secvenţial de intrare**, se apelează procedura `reset(`*f*`)`. Componenta curentă se citeşte din fişier printr-un apel de forma:

`read(`*f*, *v*`)`.

O instrucţiune de forma

`read(`*f*, v_1, v_2, ..., v_n`)`

este echivalentă cu secvenţa de instrucţiuni

`read(`*f*, v_1`); read(`*f*, v_2`); ..., read(`*f*, v_n`)`.

Sfârşitul de fişier este semnalizat de funcţia booleană `eof(`*f*`)`, care ia valoarea `true` după citirea ultimei componente.

Exemplu:

```
Program P90;
{ Citirea unui fisier cu componente de tipul Elev }
type Elev = record
              Nume : string;
              Prenume : string;
              NotaMedie : real;
            end;
     FisierElevi = file of Elev;
var FE : FisierElevi;
    E : Elev;
    str : string;
begin
  write('Dati numele fisierului de citit:  ');
  readln(str);

  assign(FE, str);   { asociaza FE cu numele din str }
  reset(FE);         { deschide fisierul FE pentru citire }

  while not eof(FE) do
    begin
      { citeste E din fisierul FE }
      read(FE, E);
      { afiseaza E pe ecran }
      writeln(E.Nume, ' ', E.Prenume, ':',
              E.NotaMedie : 5:2);
    end;
  close(FE);          { inchide fisierul FE }
  readln;
end.
```

Subliniem faptul că elaborarea programelor de prelucrare a fişierelor secvenţiale este posibilă fără a cunoaşte numărul de componente ale fişierelor respective. Teoretic, numărul de componente ce pot fi scrise într-un fişier secvenţial de ieşire

este nelimitat. Practic însă, acest număr este limitat de capacitatea de memorare a suportului extern de informații. În cazul fișierelor de intrare, programul trebuie să termine citirea consecutivă a componentelor unui fișier secvențial imediat cum se ajunge la elementul *EOF*.

C++

În cazul unui fișier secvențial, componentele acestuia pot fi accesate numai una după alta, începând cu prima. Programul ce urmează ilustrează modul de creare a unui fișier secvențial.

```
// Crearea unui fisier secvential
#include<iostream>   //pentru utilizare cout
#include<fstream>    //pentru utilizare ofstream
using namespace std;
int main()
{
 ofstream f(„test.out”);   // deschide fișier pentru scriere
 f << „Primul meu fisier!” << endl; //scrie in fisier
 cout << „Fisierul a fost creat” << endl;
 return 0;
}
```

După execuția acestui program, pe monitor va apare mesajul *Fisierul a fost creat*, iar în dosarul proiectului vom găsi fișierul `test.out` pe care îl putem citi cu orice editor de texte, de exemplu, cu aplicația *Notepad* din sistemul de operare *Windows*. De asemenea, fișierul nou-creat poate fi deschis și cu ajutorul comenzii `Open` din meniul `File` al mediului de dezvoltare a programelor C++. Evident, fișierul în cauză va conține textul *Primul meu fisier!*.

După cum s-a accentuat în paragraful precedent, în limbajul C++ fișierele sunt tratate ca secvențe de octeți. Prin urmare, gruparea datelor din fișiere în unități lexicale (numere întregi, numere reale, șiruri de caractere etc.) cade în sarcina programatorului. Pentru o înțelegere mai profundă a acestei abordări, în calitate de exemplu vom examina următoarea problemă practică, frecvent întâlnită în viața cotidiană.

Problemă: Pentru fiecare dintre elevii unei clase sunt cunoscute următoarele date: numele, prenumele, nota medie la o anumită disciplină școlară. Se cere ca datele respective să fie stocate într-un fișier secvențial și afișate pe ecran.

Rezolvare: În programul ce urmează datele respective sunt citite de la tastatură, se memorează în tabloul `e[50]` și, totodată, sunt stocate în fișierul `elevi.in`.

```
// Program P90
// Crearea fisierului ce contine date despre elevi
#include <iostream>
#include <fstream>
#include <cstring>
using namespace std;
```

```
int main()
{
    ofstream f(„elevi.in”);
    struct elev
    {
      char nume [15];
      char prenume[20];
      float NotaMedie;
    };
 int i,n;
  elev e[50];
  cout<<"Dati numarul de elevi";
  cin>>n;
  cin.get();
  for (i=1;i<=n;i++)
    {
      cout<<"Introduceti numele elevului "<<i<<" ";
      cin.get(e[i].nume,15);
      cin.get();
      cout<<"Introduceti prenumele elevului "<<i<<" ";
      cin.get(e[i].prenume,20);
      cin.get();
      cout<<"Introduceti nota medie ";
      cin>>e[i].NotaMedie;
      cin.get();
      f<<e[i].nume<<' '<<e[i].prenume<<' '<<e[i].NotaMedie<<endl;
    }
 f.close();
 return 0;
}
```

În acest program, extragerea numelui şi prenumelui elevului curent din fluxul de intrare `cin` sau, cu alte cuvinte, citirea valorilor variabilelor `Nume` şi `Prenume` de la tastatură se efectuează cu ajutorul funcției `get`. Dacă lista de argumente din apelul acestei funcții nu este vidă, ea extrage (citeşte) din fluxul supus prelucrării un şir de caractere şi îl depune în variabila respectivă. În cazul unui apel fără argumente, funcția `get` pur şi simplu trece (mută cursorul) peste un caracter, în cazul nostru – peste caracterul special *EOL*. Amintim că caracterele *EOL* apar în fluxul `cin` la acționarea tastei <*ENTER*>.

Valoarea curentă a variabilei `NotaMedie` se extrage din fluxul de intrare `cin` cu ajutorul operatorului `>>`. Acest operator citeşte, începând cu poziția curentă a cursorului, un număr real din fluxul supus prelucrării şi îl depune în variabila `NotaMedie`. După citire, cursorul va fi mutat după ultima cifră a numărului respectiv, în cazul nostru, în fața simbolului *EOL*. Pentru a muta cursorul peste acest simbol, se apelează din nou, evident, fără argumente, funcția `get`.

După citirea de la tastatură, datele despre elevul curent sunt incluse (scrise) în fluxul de ieşire `f`, care este asociat cu fişierul extern `elevi.in`.

Întrucât în limbajul C++ orice flux reprezintă doar o succesiune de caractere, fără a avea o anumită structură, simpla includere a şirurilor de caractere `Nume`, `Prenume` şi a numărului real `NotaMedie` într-un flux de ieşire nu va fi suficientă. Problema constă în faptul că orice program care va citi fişierul respectiv nu va şti unde se termină şirul de caractere `Nume` şi unde începe şirul de caractere `Prenume`. Pentru a rezolva această problemă, în programul de mai sus, în fluxul de ieşire, după fiecare din valorile `Nume`, `Prenume`, se include câte un spaţiu, iar după fiecare din valorile `NotaMedie` – câte un caracter special *EOL* (sfârşit de linie).

Evident, după terminarea procesului de execuţie a programului `P90`, datele din tabloul `e[5]` vor fi pierdute, pe când cele din fişierul nou-creat `elevi.in` vor fi păstrate pe suportul extern de informaţii. Citirea datelor din acest fişier şi afişarea lor la ecran poate fi făcută cu ajutorul programului ce urmează.

```
// Program P91
// Citirea fisierului ce contine date despre elevi
#include <iostream>
#include <fstream>
using namespace std;
int main()
{
  ifstream f(„elevi.in”); //se deschide fisierul „elevi.in”
  struct elev
   {
    char nume [15];
    char prenume[20];
    float NotaMedie;
   };

  elev e;
  f>>e.nume>>e.prenume>>e.NotaMedie;
  while (!f.eof())
   {
     cout<<e.nume<<’ ,<<e.prenume<<’ ,<<e.NotaMedie<<endl;
     f>>e.nume>>e.prenume>>e.NotaMedie;
   }
  f.close();  //se inchide fisierul
  return 0;
}
```

Datele din fluxul de intrare `f` sunt extrase (citite) până când funcţia `eof` nu va semnala faptul că s-a ajuns la caracterul *EOF*, adică la sfârşitul fişierului extern `elevi.in`.

Subliniem faptul că numărul de componente ale unui fişier nu este cunoscut din declaraţia tipului respectiv. Teoretic, într-un fişier secvenţial de ieşire pot fi înscrise un număr infinit de componente. Practic, numărul componentelor este limitat de capacitatea de memorare a suportului extern de informaţie. Citirea consecutivă a componentelor unui fişier secvenţial de intrare se încheie când se ajunge la elementul *EOF*.

Întrebări şi exerciţii

❶ Câte componente poate avea un fişier? În ce ordine se scriu/citesc componentele unui fişier secvenţial?

❷ EXERSEAZĂ! Se consideră următoarele tipuri de date:

PASCAL

```
type Data = record
              Ziua : 1..31;
              Luna : 1..12;
              Anul : integer;
            end;
     Persoana = record
          NumePrenume : string;
          DataNasterii : Data;
                end;
     FisierPersoane = file of
Persoana;
```

C++

```
struct Data
    {
     int Ziua, Luna, Anul;
    };
struct elev
    {
     char NumePrenume [40];
     Data DataNasterii;
    };
```

Elaboraţi un program care citeşte de la tastatură datele referitoare la *n* persoane şi le înscrie într-un fişier. Creaţi fişierele `FILE1.PRS`, `FILE2.PRS`, `FILE3.PRS` care trebuie să conţină datele referitoare, respectiv, la 2, 7 şi 10 persoane.

❸ REZOLVĂ! Elaboraţi un program care citeşte fişiere create de programul din exerciţiul precedent şi afişează pe ecran:
a) toate persoanele din fişier;
b) persoanele născute în anul *a*;
c) persoanele născute pe data *z.l.a*;
d) persoana cea mai în vârstă;
e) persoana cea mai tânără.

❹ REZOLVĂ! Elaboraţi un program care afişează pe ecran media aritmetică a numerelor înscrise într-un fişier de numere reale.

❺ REZOLVĂ! Într-un fişier sunt înscrise caractere arbitrare. Elaboraţi un program care afişează pe ecran numărul vocalelor din fişier.

❻ OBSERVĂ! Comentaţi următorul program:

PASCAL

```
Program P92;
{ Eroare }
type
    FisierNumere = file of
integer;
var FN : FisierNumere;
    i : integer;
    r : real;
    s : string;
begin
  Writeln('Daţi numele
       fisierului:  ');
  readln(s);
```

C++

```
// Program P92
// Eroare
#include <iostream>
#include <fstream>
using namespace std;
int main()
{
  char s[10];
  cout<<"Dati numele fisierului
       : \n";
  cin>>s;
  ifstream FN(s);
  int i;
```

```
  assign(FN, s);
  rewrite(FN);
  i:=1;
  write(FN, i);
  i:=10;
  write(FN, i);
  r:=20;
  write(FN, r);
  close(FN);
end.
```

```
  float r;
  i=1;
  FN<<i;
  i=10;
  FN<<i;
  r=20;
  FN<<r;
  FN.close();
  return 0;
}
```

1.9. Fişiere *text*

E cunoscut faptul că datele fişierelor logice din programele scrise într-un limbaj de nivel înalt se reprezintă prin secvenţe de cifre binare. Acest mod de reprezentare a datelor este convenabil în cazul memoriilor externe (discurile şi benzile magnetice, discurile optice, memoriile de tip *flash* ş.a.). În cazul echipamentelor de intrare/ieşire (tastatura, ecranul, imprimanta etc.), datele respective trebuie reprezentate într-o formă înţeleasă de om, şi anume, prin secvenţe de caractere.

Pentru a facilita interacţiunea dintre om şi sistemul de calcul, în limbajele de nivel înalt informaţia destinată utilizatorului se reprezintă în formă de fişiere ***text***. Un fişier *text* este format dintr-o secvenţă de caractere divizată în linii (*fig. 1.9*). Lungimea liniilor este variabilă. Sfârşitul fiecărei linii este indicat de un element special, notat *EOL* (*End Of Line* – sfârşit de linie). Întrucât lungimea liniilor este variabilă, poziţia unei linii în cadrul oricărui fişier *text* nu poate fi calculată din timp. În consecinţă, accesul la componentele fişierelor *text* este **secvenţial**. Cu alte cuvinte, scrierea sau citirea unei componente, adică a unui caracter sau a unei linii, este posibilă doar după ce a fost citită sau scrisă componenta precedentă.

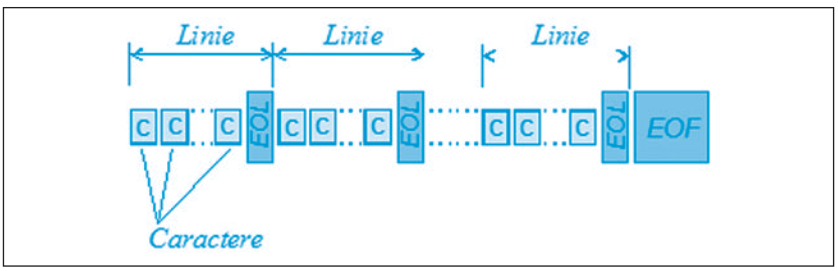


Fig. 1.9. Structura unui fişier *text*

În cadrul disciplinei şcolare Informatica se studiază următoarele modalităţi de scriere/citire a fişierelor *text*:

– la nivel de caractere;
– la nivel de lexeme;
– la nivel de linii.

În cazul **prelucrărilor la nivel de caractere**, datele din fişierul extern sunt citite/scrise caracter după caracter, fiecare dintre caractere fiind păstrat într-o variabilă de tip `char`. Asamblarea acestor caractere în numere sau şiruri de caractere cade în sarcina programatorului.

De asemenea, tot programatorul trebuie să aibă grijă de prelucrarea elementelor speciale *EOL* (sfârşit de linie).

În cazul **prelucrărilor la nivel de lexeme**, scrierea şi citirea datelor se efectuează cu ajutorul unor funcţii şi proceduri care „asamblează" datele din fişierul extern în numere şi şiruri şi le păstrează în variabile întregi, reale, caracteriale sau şiruri de caractere, după caz. Evident, lexemele în cauză trebuie să fie delimitate prin caractere albe (spaţiu, tabulator sau sfârşit de linie).

De exemplu, presupunem că fişierul de intrare conţine următoarele date:

```
82 3.14 ACCEPTAT<EOL><EOF>
```

Intuitiv, este clar că fişierul de intrare conţine numărul întreg 82, numărul real 3,14 şi şirul de caractere *ACCEPTAT*.

Prelucrarea acestui fişier la nivel de caractere constă în citirea consecutivă a fiecăruia dintre caracterele ce se conţin în el:

```
8 2   3 . 1 4   A C C E P T A T
```

şi depunerea lor în variabilele de tip `char` din program. În continuare, programatorul va trebui să scrie suplimentar o secvenţă de instrucţiuni care transformă caracterele citite în valorile de care el are nevoie.

Prelucrarea aceluiaşi fişier la nivel de lexeme constă în citirea caracterelor din fişierul de intrare şi gruparea acestora în trei lexeme:

```
82 3.14 ACCEPTAT
```

Ulterior, lexemele astfel obţinute sunt în mod automat transformate în reprezentarea internă şi depuse în variabilele de tipul respectiv: `integer`/**`int`**, `real`/**`float`** şi *şir de caractere*.

În cazul **prelucrărilor la nivel de linii**, datele din fişierul extern sunt citite/scrise linie după linie, fiecare din linii fiind păstrate într-o variabilă de tip *şir de caractere*.

În general, prelucrările la nivel de caractere sunt utile în programarea algoritmilor de procesare a textelor, de exemplu, secvenţierea în cuvinte şi propoziţii, împărţirea cuvintelor în silabe etc., iar cele de prelucrare la nivel de lexeme – în cazul programării algoritmilor de procesare a datelor preponderent numerice. Prelucrările la nivel de linii oferă utilizatorului posibilitatea să manipuleze atât cu caractere separate, cât şi cu lexeme, utilizând în acest scop funcţiile predefinite, destinate procesării şirurilor de caractere.

PASCAL

În limbajul PASCAL, un fişier *text* se defineşte printr-o declaraţie de forma:

```
var f : text;
```

în care tipul predefinit `text` fiind cunoscut oricărui program. Subliniem faptul că tipurile `text` şi **`file of`** `char` sunt distincte, întrucât fişierul **`file of`** `char`

nu include elemente *EOL*. Pentru a ne convinge de acest lucru este suficient să comparăm structurile fişierelor respective, prezentate în *figurile 1.8* şi *1.9*.

Fişierele de tip *text* pot fi prelucrate cu ajutorul procedurilor cunoscute, aplicabile oricărui tip de fişiere: `assign`, `reset`, `rewrite`, `read`, `write`, `close`. În completare, limbajul PASCAL include proceduri speciale, destinate prelucrării elementelor *EOL*:

`writeln(`*f*`)`– înscrie în fişier elementul *EOL* (sfârşit de linie);

`readln(`*f*`)`– trece la linia următoare.

Sfârşitul de linie este semnalat de funcţia booleană `eoln(`*f*`)`, care ia valoarea `true` după citirea ultimului caracter din linie.

O instrucţiune de forma

`writeln(`*f*, e_1, e_2, ..., e_n`)`

este echivalentă cu secvenţa de instrucţiuni

`write(`*f*, e_1, e_2, ..., e_n`); writeln(`*f*`)`.

O instrucţiune de forma

`readln(`*f*, v_1, v_2, ..., v_n`)`

este echivalentă cu secvenţa de instrucţiuni

`read(`*f*, v_1, v_2, ..., v_n`); readln(`*f*`)`.

Pentru introducerea şi extragerea datelor, de regulă, se utilizează fişierele *text* predefinite `Input` şi `Output`, cunoscute oricărui program PASCAL. Fişierul `Input` este destinat numai pentru operaţii de citire şi este asociat cu fişierul de intrare al sistemului de operare (de regulă, tastatura). Fişierul `Output` este destinat numai pentru operaţii de scriere şi este asociat cu fişierul standard de ieşire al sistemului de operare (de regulă, ecranul). Aceste fişiere sunt deschise şi închise automat la începutul şi, respectiv, la sfârşitul execuţiei programului. Dacă numele fişierului nu este specificat în lista parametrilor unui apel de subalgoritm, se presupune că fişierul *text* implicit este `Input` sau `Output`, în funcţie de natura subalgoritmului. De exemplu, `read(c)` este echivalent cu `read(Input,c)`, iar `write(c)` este echivalent cu `write(Output,c)`.

Pentru exemplificare, prezentăm în continuare un program care creează fişierul extern `LISTA.TXT` ce conţine numele şi prenumele a n elevi, numărul acestora nefiind cunoscut la momentul scrierii programului.

```
Program P93;
{ Crearea fisierului text LISTA.TXT prin }
 { prelucrari la nivel de linii          }
var F : text;          { fisierul logic }
    n : integer;       { numarul de elevi }
    NP : string;       { numele si prenumele elevului }
    i : integer;       { contor ciclu }
```

```
begin
  { asocierea fisierului logic F cu fisierul extern }
  assign(F, 'LISTA.TXT');
  { Deschiderea fisierului F pentru scriere }
  rewrite(F);
  { Citirea numarului de elevi de la tastatura }
  write('Dati numarul de elevi n=');
  readln(n);
  { Scrierea numarului de elevi in fisierul F }
  writeln(F, n);
  { ciclu pe numarul de elevi }
  for i:=1 to n do
    begin
      { Citirea numelui si prenumelui de la tastatura }
     writeln('Dati numele si prenumele elevului ', i, ':');
      readln(NP);
      {Scrierea numelui si prenumelui in fisierul F }
      writeln(F, NP);
    end;
  { Inchiderea fisierului F }
  close(F);
  writeln('Ati introdus ', n, ' elevi');
  readln;
end.
```

În continuarea exemplului precedent, prezentăm mai jos un program care citeşte conţinutul fişierului extern `LISTA.TXT` şi îl afişează pe ecran.

```
Program P94;
{ Citirea fisierului text LISTA.TXT prin  }
 { prelucrari la nivel de linii            }
var F : text;         { fisierul logic F    }
    n : integer;      { numarul de elevi    }
    NP : string;      { numele si prenumele }
    i : integer;      { contor ciclu        }
begin
  { Asocierea fisierului logic F cu fisierul extern }
  assign(F, 'LISTA.TXT');
  { Deschiderea fisierului F pentru citire }
  reset(F);
  { Citirea numarului de elevi n din fisierul F  }
  readln(F, n);
  { Afisarea numarului de elevi n la ecran }
  writeln('Numarul de elevi n=', n);
  { Ciclu pe numarul de elevi }
  for i:=1 to n do
```

```
    begin
      readln(F, NP); { citirea NP din fisierul F }
      writeln(NP);   { afisarea NP la ecran      }
    end;
  close(F);          { inchiderea fisierului F   }
  readln;
end.
```

Prelucrări la nivel de caractere

Fişierele *text* pot fi citite şi scrise caracter cu caracter. În acest scop, variabila *v* din apelurile `read`(*f*, *v*) şi `write`(*f*, *v*) trebuie să fie de tip `char`. Evident, prelucrarea caracterelor speciale *EOL* şi *EOF* cade în sarcina programatorului.

Pentru exemplificare, prezentăm programul `P95`, care creează pe discul curent fişierul text `FILE.TXT`. şi depune în el liniile citite de la tastatură. La momentul scrierii programului, lungimea fiecăreia dintre linii şi numărul acestora nu se cunoaşte. La introducerea datelor de la tastatură, sfârşitul liniei curente se indică prin acţionarea tastei <*ENTER*>, iar sfârşitul fişierului prin acţionarea tastelor <*CTRL+Z*>, <*ENTER*>.

```
Program P95;
{ Crearea fisierului text FILE.TXT prin prelucrari   }
{ la nivel de caractere                               }
var F : text;            { fisierul logic F          }
    C : char;
begin
  assign(F, 'FILE.TXT'); { asociaza F cu FILE.TXT    }
  rewrite(F);            { deschide F pentru scriere }
  while not eof do       { verifica daca-i <CTRL+Z>  }
    begin
      while not eoln do  { verifica daca-i <ENTER>   }
        begin
          read(C);       { citeste C de la tastatura }
          write(F, C);   { scrie C in fisierul F     }
        end;
      writeln(F);        { scrie EOL in fiserul F    }
      readln;            { trece la linia urmatoare  }
    end;
  close(F);              { scrie EOF in fisierul F   }
end.
```

Accentuăm faptul că în programul de mai sus componentele fişierului de intrare `Input` (tastatura) se citesc la nivel de caractere, iar sfârşitul fiecăreia dintre linii (elementul special *EOL*) se depistează cu ajutorul funcţiei `eoln`. Într-un mod similar, sfârşitul fişierului de intrare (elementul special *EOF*) se depistează cu ajutorul funcţiei `eof`.

Programul ce urmează afişează conţinutul fişierului extern `FILE.TXT` pe ecran.

```
Program P96;
{ Citirea fisierului text FILE.TXT prin prelucrari }
{ la nivel de caractere                            }
var F : text;              { fisierul logic F      }
    C : char;
begin
  assign(F, 'FILE.TXT');   { asociaza F cu FILE.TXT   }
  reset(F);                { deschide F pentru citire }
  while not eof(F) do      { testeaza daca-i EOF in F }
    begin
      while not eoln(F) do { testeaza daca-i EOL in F }
        begin;
          read(F, C);      { citeste C din F          }
          write(C);        { afiseaza C la ecran      }
        end;
      readln(F);           { in F trece la linia urmatoare  }
      writeln;             { pe ecran trece la o linie noua }
    end;
  close(F);                { inchide fisierul F }
  readln;
end.
```

În programul de mai sus, componentele fişierului de ieşire `Output` (ecranul) se scriu la nivel de caractere. Sfârşitul *EOL* al fiecăreia dintre linii se înscrie prin apelul procedurii `readln`, iar sfârşitul de fişier *EOF* – prin apelul procedurii `close`.

Prelucrări la nivel de lexeme

Explorarea caracter cu caracter a fişierelor *text* este greoaie în situaţia în care secvenţele de caractere din text trebuie interpretate ca formând date de tip `integer`, `real`, `boolean`, *şir de caractere*. Conversia între forma externă şi reprezentarea internă a acestor tipuri ar cădea în sarcina programatorului. Pentru a uşura munca acestuia, procedurile de citire/înscriere au fost extinse în felul următor.

În cazul fişierelor *text*, variabila *v* dintr-un apel `read`(*f*, *v*) poate fi de tipul `integer`, `real`, `char` sau *şir de caractere*. La citire, secvenţa de caractere care reprezintă variabila *v* va fi transformată în reprezentarea internă.

Expresia *e* dintr-un apel `write`(*f*, *e*) poate fi urmată de specificatori de format. Valoarea expresiei poate fi de tipul `integer`, `real`, `boolean`, `char` sau *şir de caractere*. La scriere, valoarea respectivă este transformată din reprezentarea internă într-o secvenţă de caractere.

Secvenţele de caractere citite/scrise de procedurile `read/write` se conformează sintaxei constantelor de tipul variabilei/expresiei *v*/*e*.

Pentru exemplificare, prezentăm programul `P97` care:

1) citeşte de la tastatură o linie ce conţine numere reale a, b, c – lungimile laturilor unui triunghi;

2) scrie numerele citite în linia curentă a fişierului *text* `IN.TXT`;

3) repetă paşii 1-2 până când utilizatorul nu va tasta <*CTRL+Z* > <*ENTER*>;

4) citeşte din linia curentă a fişierului *text* `IN.TXT` numerele reale *a*, *b*, *c*;

5) calculează semiperimetrul *p* şi aria *s* ale triunghiului respectiv;

6) scrie numerele reale *a*, *b*, *c*, *p* , *s* în fişierul *text* `OUT.TXT`;
7) repetă paşii 4-6 până când va fi citită ultima linie din fişierul *text* `OUT.TXT`;
8) afişează conținutul fişierului `OUT.TXT` pe ecran.

```
Program P97;
{ Prelucrarea triunghiurilor                }
{ Scrierea si citirea la nivel de lexeme    }
var F, G : text;          { fisiere logice  }
    a, b, c, p, s : real;
    Str : string;
begin

  { Citirea datelor de la tastatura si stocarea lor in F}
  assign(F, 'IN.TXT');  { asociaza F cu IN.TXT          }
  rewrite(F);           { deschide F pentru scriere     }
  writeln('Dati numerele reale a, b, c:');
  while not eof do
    begin
      readln(a, b, c);  { citeste a, b, c de la tastatura   }
      writeln(F, a:8:2, b:8:2, c:8:2); { scrie a, b, c in F }
    end;
  close(F);             { inchide F }

  { Citirea datelor din F, efectuarea calculelor si }
  { stocarea rezultatelor in G                      }
  reset(F);             { deschide F pentru citire  }
  assign(G, 'OUT.TXT'); { asociaza G cu OUT.TXT     }
  rewrite(G);           { deschide G pentru scriere }
  while not eof(F) do
    begin
      readln(F, a, b, c);  { citeste a, b, c din F }
      write(G, a:8:2, b:8:2, c:8:2); { scrie a, b, c in G }
      p:=(a+b+c)/2;
      s:=sqrt(p*(p-a)*(p-b)*(p-c));
      writeln(G, p:15:2, s:15:4); { scrie p, s în G }
    end;
  close(F);             { inchide F }
  close(G);             { inchide G }

  { Citirea rezultatelor din G si afisarea lor la ecran }
  reset(G);             { deschide G pentru citire }
  while not eof(G) do
    begin
      readln(G, Str);   { citeste Str din G        }
      writeln(Str);     { afiseaza Str pe ecran    }
    end;
  close(G);             { inchide G              }
  readln;
end.
```

Pentru datele de intrare

```
1 1 1 <ENTER>
3 4 6 <ENTER>
<CTRL+Z ><ENTER>
```

programul P97 afişează pe ecran:

```
1.00 1.00 1.00 1.50 0.4330
3.00 4.00 6.00 6.50 5.3327
```

Accentuăm faptul că programul de mai sus este destinat pentru prelucrarea datelor referitoare la un număr necunoscut de triunghiuri, fapt ce-i oferă o flexibilitate mai mare în utilizare. Această flexibilitate se datorează utilizării funcției `eof`. Cu ajutorul acestei funcții se testează dacă în procesul citirii s-a ajuns la sfârşitul fiecăruia dintre fişierele `Input` (tastatura) şi `IN.TXT`.

C++

În limbajul C++, un fişier text conține numai caractere ASCII organizate pe linii de lungimi diferite. Elementul *EOL* se reprezintă prin caracterul *escape* "`\n`" (*new line*, linie nouă). Sfârşitul fişierului este specificat prin elementul *EOF*. Întrucât lungimea liniilor este variabilă, poziția unei linii în cadrul fişierului nu poate fi calculată din timp. În consecință, accesul la componentele fişierelor *text* este secvențial.

Întrucât limbajul C++ nu impune nicio structură asupra conținutului fişierelor, secvențierea lor în lexeme (numere, şiruri de caractere) şi linii cade în sarcina programatorului. În acest scop pot fi folosite toate funcțiile destinate prelucrării fişierelor secvențiale studiate în paragraful 1.8.

Pentru exemplificare, prezentăm în continuare programul P93, care creează pe discul curent un fişier *text*, numele căruia este dat de la tastatură. Liniile fişierului sunt delimitate prin acționarea tastei <*ENTER*>, iar sfârşitul fişierului – prin acționarea tastelor <*CTRL+Z* > <*ENTER*>.

```
// Program P93
// Crearea unui fişier text
#include <iostream>
#include <fstream>
using namespace std;
int main()
{
  char nume_fisier[80];
  char c;
  cout << "\n\tProgramul creeaza un fisier text\n";
  cout << "\n\tIntroduceti numele fisierului: ";
  cin.getline(nume_fisier,80);
  ofstream f(nume_fisier);
  cout << "\n\tIntroduceti caractere.Sfarsit Enter CTRL+Z\n";
  while(cin.get(c))
       f << c;
  f.close();
  return 0;
}
```

Prelucrări la nivel de caractere

Reamintim că la citirea cu operatorul >> sunt ignorate caracterele albe (spațiul, tabulatorul, sfârşitul de linie). Prin urmare, dacă intenționăm să citim unul după altul toate caracterele din fişier, inclusiv cele albe, vom utiliza funcția `get()`.

```
// Program P94
// Citirea dintr-un fisier text
#include <iostream>
#include <fstream>
using namespace std;
int main()
{
  char nume_fisier[80];
  char c ;
  cout << "\n\tProgramul citeste un fisier text\n";
  cout << "\n\tIntroduceti numele fisierului: ";
  cin.getline(nume_fisier,80);
  ifstream f(nume_fisier);
  f.get(c);
  while(!f.eof())
    {
     f.get(c);
     cout<<c;
    }
f.close();
return 0;
}
```

Pentru a verifica dacă poziția curentă de citire/scriere a ajuns la sfârşitul unui fişier, se foloseşte funcția `eof()`, care întoarce valoarea 0 dacă poziția curentă nu este la sfârşitul fişierului şi o valoare diferită de 0 dacă poziția actuală indică sfârşitul de fişier.

Programul de mai jos copiază caracter cu caracter datele din fişierul *text* `IN.TXT` în fişierul *text* `OUT.TXT`.

```
// Program P95
// Copierea caracter cu caracter a fisierelor text
#include <iostream>
#include <fstream>
using namespace std;
int main()
{
char c ;
ifstream f("in.txt");    // fluxul f se asociaza cu in.txt
ofstream g("out.txt");   // fluxul g se asociaza cu out.txt
f.get(c);                // din f se citeste primul caracter
while(!f.eof())          // ciclu pe caracterele din f
    {
```

```
      g.put(c);           // caracterul citit se scrie in g
     f.get(c);          // din f se citeste caracterul urmator
     }
f.close();               // inchiderea fisierului in.txt
g.close();               // inchiderea fisierului out.txt
return 0;
}
```

Prelucrări la nivel de linii

În cazul în care şirurile de caractere din fişierul de intrare, inclusiv caracterele albe, trebuie citite linie cu linie, se utilizează funcţia `getline`.

Programul de mai jos copiază linie cu linie datele din fişierul *text* `IN.TXT` în fişierul *text* `OUT.TXT`.

```
// Program P96
// Copierea linie cu linie a fisierelor text
#include <iostream>
#include <fstream>
using namespace std;
int main()
{
char linie[80];
ifstream f("in.txt");      // fluxul f se asociaza cu in.txt
ofstream g("out.txt");     // fluxul g se asociaza cu out.txt
while(!f.eof())            // ciclu pe liniile din f
    {
     f.getline(linie,80); // din f se citeste linia curenta
     g<<linie<<endl;      // linia citita este scrisa in g
    }
f.close();                 // inchiderea fisierului in.txt
g.close();                 // inchiderea fisierului out.txt

return 0;
}
```

Prelucrări la nivel de lexeme

Programul ce urmează utilizează funcţiile destinate prelucrării fişierelor C++ pentru secvenţierea acestora în lexeme. Mai exact, acest program:

1) citeşte de la tastatură o linie ce conţine numere reale *a*, *b*, *c* – lungimile laturilor unui triunghi;
2) scrie numerele citite în linia curentă a fişierului *text* `in.txt`;
3) repetă paşii 1-2 până când utilizatorul nu va tasta <*CTRL+Z* > <*ENTER*>;
4) citeşte din linia curentă a fişierului *text* `in.txt` numerele reale *a*, *b*, *c*;
5) calculează semiperimetrul *p* şi aria *s* ale triunghiului respectiv;
6) scrie numerele reale *a*, *b*, *c*, *p*, *s* în fişierul *text* `out.txt`;

7) repetă paşii 4-6 până când nu va fi citită ultima linie din fişierul *text* `out.txt`;
8) afişează conținutul fişierului `out.txt` pe ecran.

```
//Program P97
//Prelucrarea triunghiurilor
//Scrierea si citirea la nivel de lexeme
#include <iostream>
#include <iomanip>
#include <cmath>
#include <fstream>
using namespace std;
int main()
{
  float a,b,c;
  float p,s;
  int i,n;
  char linie[80];

  //Citirea datelor de la tastatura si stocarea lor in f
  ofstream f("in.txt");  //deschide f pentru scriere
  cout<<"Dati numerele reale a,b,c \n";
  cin>>a>>b>>c;          // citeste a, b, c de la tastatura
  while (!cin.eof())     //repeta pana la tastarea <CTRL+Z >
    <ENTER>
   {
      f<<a<<" "<<b<<" "<<c<<endl;  // scrie a, b, c in f
      cin>>a>>b>>c;      // citeste a, b, c de la tastatura
   }
  f.close();             // inchide f

  //Citirea datelor din f, efectuarea calculelor si
  // stocarea rezultatelor in g
  ifstream f("in.txt");    // deschide f pentru citire
  ofstream g("out.txt");   // deschide g pentru scriere
  f>>a>>b>>c;              // citeste a,b,c din f
  while(!f.eof())
   {
   g<<a<<" "<<b<<" "<<c<<"  "; // scrie a,b,c in g
   p=(a+b+c)/2;
   s=sqrt(p*(p-a)*(p-b)*(p-c));
   g<<p<<"  "<<s<<"\n";  // scrie p si s in g
   f>>a>>b>>c;           // citeste a,b,c din f
 }
 f.close();              // inchide f
 g.close();              // inchide g
```

```
// Citirea rezultatelor din g si afisarea lor la ecran
ifstream g("out.txt");   // deschide g pentru citire
while(!g.eof())
 {
  g.getline(linie,80);   // citeste din g linia curenta
  cout<<linie<<endl;     // afiseaza linia citita la ecran
 }
g.close();              // inchide g
return 0;
}
```

Pentru datele de intrare

```
1 1 1 <ENTER>
3 4 6 <ENTER>
<CTRL+Z ><ENTER>
```

programul P97 afişează pe ecran:

```
1 1 1  1.5  0.433013
3 4 6  6.5  5.332684
```

Accentuăm faptul că programul de mai sus este destinat pentru prelucrarea datelor referitoare la un număr necunoscut de triunghiuri, fapt ce-i oferă o flexibilitate mai mare în utilizare. Această flexibilitate se datorează utilizării funcției `eof`. Cu ajutorul acestei funcții se testează dacă în procesul citirii s-a ajuns la sfârşitul fiecăruia dintre fişierele `Input` (tastatura) şi `IN.TXT`.

Întrebări şi exerciţii

❶ EXERSEAZĂ! Desenați o schiță care redă structura unui fişier *text*. Explicați destinația fiecărui element grafic al schiței respective.

❷ ANALIZEAZĂ! Se consideră următorul fişier text de intrare:

```
Date 9.8 12 23.67 STOP<EOL><EOF>
```

Explicați cum vor fi citite datele din acest fişier şi ce transformări vor avea loc în cazul următoarelor prelucrări: (*a*) la nivel de caractere; (*b*) la nivel de lexeme.

❸ ANALIZEAZĂ! Se consideră următorul fişier *text* de intrare:

```
5.1       9.3         Admis<EOL>6.4     4.3
Respins<EOL><EOF>
```

Explicați cum vor fi citite datele din acest fişier şi ce transformări vor avea loc în cazul următoarelor prelucrări: (*a*) la nivel de caractere; (*b*) la nivel de lexeme.

❹ CREEAZĂ! Dați exemple de fişiere de intrare ce conțin datele inițiale ale unei probleme din fizică şi ale alteia din biologie şi descrieți procesul de citire a datelor din ele la nivel de caractere şi la nivel de lexeme.

❺ ANALIZEAZĂ! Care este diferența dintre prelucrarea unui fișier *text* la nivel de caractere și prelucrarea aceluiași fișier la nivel de lexeme? Dați exemple de algoritmi ce necesită implementarea prin prelucrări la nivel de caractere și prelucrări la nivel de lexeme.

❻ ÎNVAȚĂ SĂ ÎNVEȚI! Pornind de la cele două modalități de citire a fișierelor de intrare la nivel de caractere și la nivel de lexeme, explicați modalitățile respective în cazul fișierelor *text* de ieșire. Dați exemple.

❼ EXPLICĂ! Explicați semnificația elementelor *EOL* și *EOF*. Cum sunt prelucrate aceste elemente?

❽ EXPLICĂ! Explicați cum se efectuează scrierea și citirea datelor în cazul fișierelor *text*. Care proceduri/funcții sunt utilizate în acest scop?

❾ OBSERVĂ! Lansați în execuție următorul program:

PASCAL

```
Program P98;
{ Asocierea fișierului FN cu
consola }
type FisierNumere = file of in-
teger;
var FN : FisierNumere;
    i : integer;
begin
  assign(FN, 'CON');
  rewrite(FN);
  i:=1;
  write(FN, i);
  i:=2;
  write(FN, i);
  i:=3;
  write(FN, i);
  close(FN);
  readln;
end.
```

C++

```
// Program P98
// Asocierea fișierului FN
// cu consola
#include <iostream>
#include <fstream>
using namespace std;
int main()
{
  int i;
  ofstream FN („CON”);
  i=1;
  FN<<i;
  i=2;
  FN<<i;
  i=3;
  FN<<i;
  FN.close();
  return 0;
}
```

Explicați rezultatele afișate pe ecran.

❿ ANALIZEAZĂ! Se consideră fișierul logic `f`. Care din următoarele instrucțiuni destinate depistării sfârșitului de fișier sunt corecte?

PASCAL

a)
```
if eoln(f) then
     write ('Sfarsit de fisier')
else write ('Nu-i sfarsit de fisier');
```

b)
```
if eof(f) then
     write ('Sfarsit de fisier')
else write ('Nu-i sfarsit de fisier');
```

c)
```
if eoln(f)=false then
     write ('Sfarsit de fisier')
else write ('Nu-i sfarsit de fisier');
```

d)
```
if not eof(f) then
     write ('Sfarsit de fisier')
else write ('Nu-i sfarsit de fisier');
```

C++

a)
```
if (eof(f))
     cout << "Sfarsit de fisier";
else cout << "Nu-i sfarsit de fisier";
```

b)
```
if (f.eof())
     cout << "Sfarsit de fisier";
else cout << "Nu-i sfarsit de fisier";
```

c)
```
if (!eof(f))
     cout << "Sfarsit de fisier";
else cout << "Nu-i sfarsit de fisier";
```

d)
```
if (!f.eof())
     cout << "Nu-i sfarsit de fisier";
else cout << "Sfarsit de fisier";
```

⓫ APLICĂ! Elaborați un program care afișează pe ecran conținutul oricărui fișier *text*.

⓬ REZOLVĂ! Elaborați un program care afișează pe ecran numărul de vocale dintr-un fișier *text*.

⓭ PROGRAMEAZĂ! Datele de intrare ale unui program sunt înmagazinate într-un fișier *text*. Fiecare linie a fișierului conține două numere întregi și trei numere reale separate prin spații. Elaborați un program care afișează suma numerelor întregi și suma numerelor reale din fiecare linie pe ecran.

⓮ PROGRAMEAZĂ! Datele de intrare ale unui program sunt înmagazinate într-un fișier *text*. Fiecare linie a fișierului conține trei numere reale separate prin spațiu și unul din cuvintele `ADMIS`, `RESPINS`. Elaborați un program care:
a) afișează conținutul fișierului în studiu pe ecran;
b) creează o copie de rezervă a fișierului;
c) creează un fișier *text* liniile căruia conțin media celor trei numere reale din liniile respective ale fișierului de intrare;
d) afișează pe ecran liniile fișierului de intrare, precedate de numerele de ordine 1, 2, 3 ș.a.m.d.

⓯ PROGRAMEAZĂ! Fiecare linie a unui fișier *text* conține următoarele date, separate prin spații:
– numărul de ordine (număr întreg);
– numele (un șir de caractere ce nu conține spații);
– prenumele (un șir de caractere ce nu conține spații);
– nota la disciplina 1 (număr real);
– nota la disciplina 2 (număr real);
– nota la disciplina 3 (număr real).

Elaborați un program care:
a) creează o copie de rezervă a fișierului în studiu;
b) afișează conținutul fișierului pe ecran;
c) creează un fișier *text* liniile căruia conțin următoarele date separate prin spații:
– numărul de ordine (număr întreg);
– numele (un șir de caractere ce nu conține spații);
– prenumele (un șir de caractere ce nu conține spații);
– nota medie (număr real);

Fișierul creat în punctul *c* trebuie afișat pe ecran.

Capitolul 2

INFORMAȚIA

2.1. Cantitatea de informaţie

Sensul uzual al cuvântului **informaţie** „ştire, comunicare verbală, scrisă sau transmisă prin alte metode despre anumite fapte, evenimente, activităţi etc.” se concretizează într-un compartiment special al matematicii, denumit **teoria informaţiei**. Conform acestei teorii, **sursa de informaţie** se descrie printr-o variabilă S care poate lua valori dintr-o mulţime finită de elemente distincte $\{s_1, s_2, ..., s_n\}$. Se consideră că valorile curente ale variabilei S nu sunt cunoscute din timp. E cunoscută numai mulţimea $\{s_1, s_2, ..., s_n\}$, **denumită mulţimea mesajelor posibile**.

De exemplu, semaforul de circulaţie poate fi reprezentat ca o sursă de informaţie, mulţimea de mesaje posibile ale căruia este {*verde, galben, roşu*}. Aparatul de telegrafiat reprezintă o sursă de informaţie, mulţimea de mesaje posibile ale căruia include literele *A, B, C, ..., Z,* cifrele *0, 1, 2, ..., 9* şi semnele de punctuaţie. Mesajele posibile ale tastaturii sunt: „*Este acţionată tasta A*”, „*Este acţionată tasta B*”, ..., „*Este acţionată tasta F1*”, „*Este acţionată tasta F2*”, ..., „*Sunt acţionate concomitent tastele CTRL şi BREAK*” etc.

Mesajele se transmit de la sursă către destinatar printr-un mediu fizic, numit **canal de transmisie** (*fig. 2.1*). De exemplu, mesajele telegrafice se transmit prin fir, mesajele radio prin eter, mesajele tastaturii printr-un set de conductori. **Perturbaţiile** (zgomotele) din mediul fizic amintit pot altera mesajele transmise.

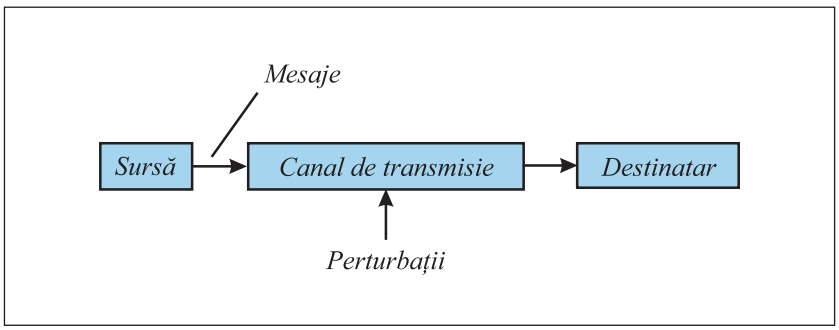


Fig. 2.1. Schema generală a unui sistem de transmisie a informaţiei

Evident, valoarea curentă a variabilei S devine cunoscută destinatarului numai după recepţionarea mesajului respectiv.

Cantitatea de informaţie I ce este conţinută într-un mesaj emis de sursă se determină din relaţia:

$$I = \log_a n,$$

unde n este numărul de mesaje posibile ale sursei. Valoarea concretă a constantei a se stabileşte prin alegerea **unității de măsură a cantității de informație**. De obicei, ca unitate de măsură se utilizează **bitul**.

Un bit este cantitatea de informație din mesajul unei surse cu numai două mesaje posibile.

Prin urmare, ca şi în cazul altor mărimi (lungimea, masa, temperatura etc.), cantitatea de informație se măsoară prin compararea cu **etalonul**. Întrucât pentru sursa-etalon $n = 2$, din ecuația:

$$\log_a 2 = 1 \quad (bit)$$

obținem $a = 2$. În consecință, cantitatea de informație I, măsurată în biți, se determină din relația:

$$I = \log_2 n \ (bit).$$

În *tabelul 2.1* sunt prezentate valorile frecvent utilizate ale funcției $\log_2 n$.

Tabelul 2.1

Valorile funcției $\log_2 n$

n	$\log_2 n$	n	$\log_2 n$
1	0,000	21	4,392
2	1,000	22	4,459
3	1,585	23	4,524
4	2,000	24	4,585
5	2,322	25	4,644
6	2,585	26	4,700
7	2,807	27	4,755
8	3,000	28	4,807
9	3,170	29	4,858
10	3,322	30	4,907
11	3,459	31	4,954
12	3,585	32	5,000
13	3,700	33	5,044
14	3,807	34	5,087
15	3,907	35	5,129
16	4,000	36	5,170
17	4,087	37	5,209
18	4,170	38	5,248
19	4,248	39	5,285
20	4,322	40	5,322

Să analizăm câteva exemple. Cantitatea de informație a unui mesaj de semafor este de

$$I = \log_2 3 \approx 1{,}585 \textit{ biți}.$$

Cantitatea de informație a unei litere a alfabetului latin {*A, B, C, ..., Z*}, $n = 26$, este de

$$I = \log_2 26 \approx 4{,}700 \textit{ biți}.$$

Cantitatea de informație a unei litere a alfabetului grec {*A, B,* Γ, Δ , ..., Ω}, $n = 24$, este de

$$I = \log_2 24 \approx 4{,}585 \textit{ biți}.$$

Dacă se cunoaşte cantitatea de informație I ce este conținută într-un mesaj, cantitatea totală de **informație emisă** de sursă se determină din relația:

$$V = N I,$$

unde N este numărul de mesaje transmise.

Cantitățile mari de informație se exprimă prin multiplii unui bit:

1 *Kilobit* (*Kbit*) = 2^{10} = 1 024 *biți* (≈10^3 *biți*);

1 *Megabit* (*Mbit*) = 2^{20} = 1 048 576 *biți* (≈10^6 *biți*);

1 *Gigabit* (*Gbit*) = $2^{30} \approx 10^9$ *biți*;

1 *Terabit* (*Tbit*) = $2^{40} \approx 10^{12}$ *biți*;

1 *Petabit* (*Pbit*) = $2^{50} \approx 10^{15}$ *biți*.

Întrebări şi exerciţii

❶ Cum se defineşte o sursă de informație? Dați câteva exemple.

❷ Care este destinația canalului de transmisie?

❸ Cum se determină cantitatea de informație dintr-un mesaj? Dar din N mesaje?

❹ Care este unitatea de măsură a informației şi ce semnificație are ea?

❺ Determinați cantitatea de informație într-un mesaj al surselor cu următoarele mesaje posibile:
a) literele mari şi mici ale alfabetului latin;
b) literele mari şi mici ale alfabetului grec;
c) literele mari şi mici ale alfabetului român;
d) cifrele zecimale 0, 1, 2, ..., 9;
e) cifrele 0, 1, 2, ..., 9, semnele +, –, ×, / şi parantezele ( );
f) indicațiile numerice de forma *hh:mm* (*hh* – ora, *mm* – minutele) ale unui ceas electronic;
g) indicațiile numerice de forma *hh:mm:ss* (*ss* – secundele) ale unui ceas electronic;
h) indicațiile numerice de forma *zz.ll.aa* (*zz* – ziua, *ll* – luna, *aa* – anul) ale unui calendar electronic.

❻ Pentru fiecare dintre sursele indicate în exercițiul 5 determinați cantitatea de informație ce este conținută în 1 000 de mesaje emise de sursă.

❼ Elaborați un program care calculează cantitatea de informație din N mesaje emise de o sursă cu n mesaje posibile.

2.2. Codificarea şi decodificarea informaţiei

Se numeşte **semn** un element al unei mulțimi finite de obiecte ce se pot distinge. O mulțime de semne ordonate liniar se numeşte **alfabet**.

Prezentăm în continuare unele dintre nenumăratele alfabete folosite de oameni:

a) alfabetul cifrelor zecimale: 0, 1, 2, ..., 9;

b) alfabetul literelor latine mari: *A, B, C, ..., Z*;

c) mulțimea semnelor zodiacului;

d) mulțimea fazelor lunii.

O importanță deosebită o au alfabetele de numai două semne. Aceste alfabete se numesc **alfabete binare**, iar semnele respective – **semne binare**.

Prezentăm câteva exemple de alfabete binare:

a) cifrele {*0, 1*};

b) perechea de culori {*roşu, galben*};

c) perechea de stări {*închis, deschis*};

d) perechea de răspunsuri {*da, nu*};

e) perechea de tensiuni {0*V*, 2*V*};

f) perechea de stări {*magnetizat, nemagnetizat*};

g) perechea de semne {+, –} etc.

S-a convenit ca semnele unui alfabet binar să fie reprezentate prin cifrele {*0, 1*}, denumite **cifre binare** (*binary digit*).

Un şir finit din m semne, dintre care unele se pot repeta, formează un **cuvânt**, m reprezentând **lungimea cuvântului**. Cuvintele formate din semne binare se numesc **cuvinte binare**. Evident, cuvintele pot avea lungime variabilă sau constantă. În ultimul caz ele se numesc **cuvinte *m*-poziționale**. În continuare prezentăm unele mulțimi de cuvinte cu lungime constantă:

1-poziționale: {0, 1};

2-poziționale: {00, 01, 10, 11};

3-poziționale: {000, 001, 010, 011, 100, 101, 110, 111};

4-poziționale: {0000, 0001, ..., 1110, 1111}.

Se observă că cuvintele (m + 1)-poziționale se formează câte două din cuvintele m-poziționale prin adăugarea cifrelor binare 0 şi 1. Prin urmare, mulțimea cuvintelor m-poziționale include 2^m cuvinte distincte.

Cuvintele binare se utilizează pentru reprezentarea, transmiterea, păstrarea şi prelucrarea mesajelor s_1, s_2, ..., s_n ale sursei de informație (*fig. 2.2*).

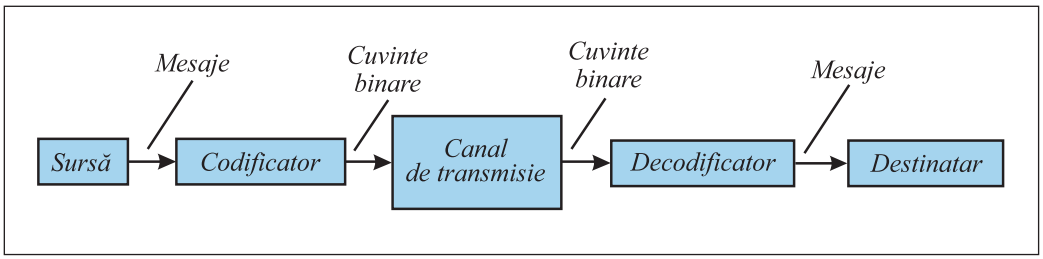


Fig. 2.2. Codificarea şi decodificarea mesajelor în sistemele de transmisie a informaţiei

Regula de transformare a mesajelor în cuvinte se numeşte cod, iar operaţia respectivă – codificare. Operaţia inversă codificării se numeşte decodificare. Dispozitivele tehnice care realizează operaţiile în cauză se numesc, respectiv, codificator şi decodificator.

Cel mai simplu este codul în care mesajelor posibile s_1, s_2, ..., s_n le corespund cuvinte binare de lungime constantă m. Acest cod denumit **cod *m*-poziţional** poate fi definit cu ajutorul unui tabel. În *figura 2.3* sunt prezentate tabelele respective pentru surse cu n = 2, 3, 4, ..., 8 mesaje posibile.

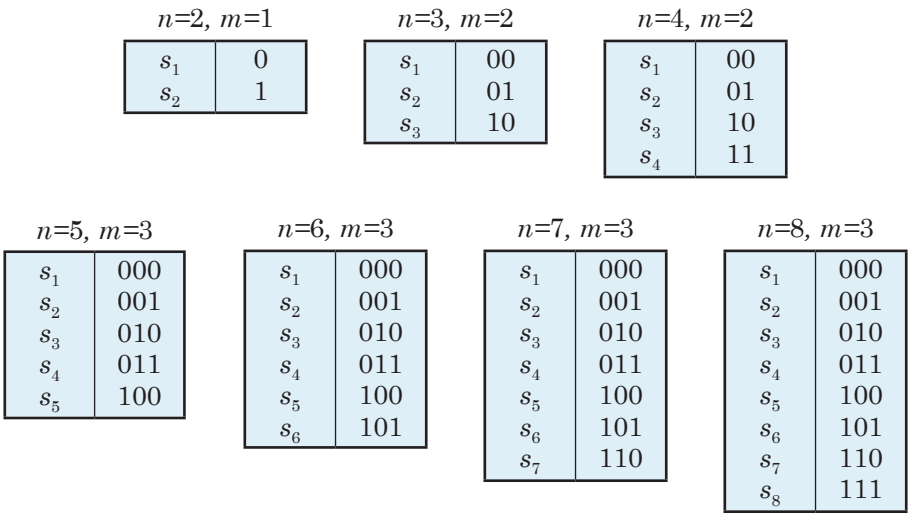


Fig. 2.3. Coduri de cuvinte cu lungime constantă (coduri *m*-poziţionale)

Operaţiile de codificare şi decodificare constau în extragerea datelor necesare din tabel. Evident, decodificarea va fi univocă numai atunci când cuvintele binare incluse în tabel sunt distincte. Acest lucru este posibil dacă lungimea m a cuvintelor de cod satisface inegalitatea

$$\mathbf{2}^{m} \geq \boldsymbol{n}.$$

După logaritmare obţinem:

$$\boldsymbol{m} \geq \mathbf{log}_2\, \boldsymbol{n}.$$

Întrucât expresia $\log_2 n$ exprimă cantitatea de informaţie, se poate afirma:

Lungimea cuvintelor unui cod poziţional trebuie să fie mai mare sau egală cu cantitatea de informaţie a unui mesaj.

De exemplu, lungimea cuvintelor pentru codificarea literelor mari ale alfabetului latin {*A, B, C, ...Z*}, n = 26, se determină din relaţia

$$\boldsymbol{m} \geq \mathbf{log}_2\, \mathbf{26} \approx \mathbf{4{,}700}.$$

Stabilind m = 5, putem forma cuvintele binare ale codului 5-poziţional:

A – 00000
B – 00001
C – 00010

D – 00011
E – 00100
...
Z – 11001.

Un astfel de cod a fost propus de filosoful şi omul de stat englez Francis Bacon încă în anul 1580.

Algoritmii de elaborare a **codurilor cu cuvinte de lungime variabilă** sunt mult mai complicaţi şi se studiază în cursurile avansate de informatică.

Întrebări şi exerciţii

❶ Ce este un alfabet? Daţi exemple de alfabete binare.

❷ Cum se reprezintă semnele oricărui alfabet binar?

❸ Explicaţi cum pot fi formate cuvintele binare (m+1)-poziţionale. Care este numărul cuvintelor binare m-poziţionale distincte?

❹ Care este destinaţia unui cod? Cum se defineşte codul m-poziţional?

❺ Cum se efectuează codificarea şi decodificarea mesajelor în cazurile când codul este definit printr-un tabel?

❻ Codificaţi mesajele s_3, s_4 şi s_6 ale unei surse cu 7 mesaje posibile. Utilizaţi codul 3-poziţional din *figura 2.3*.

❼ Decodificaţi mesajele 100, 000 şi 010, reprezentate în codul 3-poziţional din *figura 2.3*, n = 5.

❽ Cum se determină numărul de semne binare necesare pentru formarea cuvintelor de lungime constantă ale unui cod?

❾ Utilizând codul 3-poziţional din *figura 2.3*, n = 6, codificaţi şirul de mesaje s_1, s_2, s_6, s_5, s_3, s_6, s_3, s_2, s_1.

❿ Cum influenţează cantitatea de informaţie a unui mesaj asupra lungimii cuvintelor de cod?

⓫ Explicaţi sensul termenilor **cantitate de informaţie** şi **informaţie**.

⓬ Elaboraţi un program care codifică şi decodifică literele alfabetului latin. Se va utiliza codul propus de Francis Bacon.

⓭ Elaboraţi un program care alcătuieşte tabelul unui cod m-poziţional pentru o sursă cu n mesaje posibile.

2.3. Coduri frecvent utilizate

Orice cod, utilizat pentru reprezentarea, transmiterea, păstrarea şi prelucrarea informaţiei, trebuie să fie econom şi insensibil la perturbaţii, iar echipamentele respective de codificare şi decodificare – să fie simple. Pe parcursul dezvoltării tehnicii de calcul au fost elaborate mai multe coduri. Aceste coduri se clasifică în coduri numerice şi coduri alfanumerice.

Codurile numerice oferă posibilitatea reprezentării cifrelor {0, 1, 2, ..., 9} prin cuvinte binare 4-poziţionale. Exemple de coduri numerice sunt prezentate în *tabelul 2.2*, p. 98.

Tabelul 2.2

Coduri numerice

Cifra	Denumirea codului			
	Direct	***Gray***	***Aiken***	***Exces 3***
0	0000	0000	0000	0011
1	0001	0001	0001	0100
2	0010	0011	0010	0101
3	0011	0010	0011	0110
4	0100	0110	0100	0111
5	0101	0111	1011	1000
6	0110	0101	1100	1001
7	0111	0100	1101	1010
8	1000	1100	1110	1011
9	1001	1101	1111	1100

Codurile alfanumerice reprezintă prin cuvinte binare cifrele 0, 1, 2, ..., 9, literele mari şi mici ale alfabetului, semnele de punctuație, semnele operațiilor aritmetice etc. În *tabelul 2.3* este prezentat codul **ASCII** (*American Standard Code for Information Interchange*), inventat în anul 1968.

Tabelul 2.3

Codul ASCII

Simbol	Cuvânt binar	Echivalent zecimal	Simbol	Cuvânt binar	Echivalent zecimal
Spațiu	0100000	32	P	1010000	80
!	0100001	33	Q	1010001	81
"	0100010	34	R	1010010	82
#	0100011	35	S	1010011	83
$	0100100	36	T	1010100	84
%	0100101	37	U	1010101	85
&	0100110	38	V	1010110	86
'	0100111	39	W	1010111	87
(	0101000	40	X	1011000	88
)	0101001	41	Y	1011001	89
*	0101010	42	Z	1011010	90
+	0101011	43	[	1011011	91
,	0101100	44	\	1011100	92
-	0101101	45	]	1011101	93
.	0101110	46	^	1011110	94

Continuarea tab. 2.3

Simbol	Cuvânt binar	Echivalent zecimal	Simbol	Cuvânt binar	Echivalent zecimal
/	0101111	47	_	1011111	95
0	0110000	48	`	1100000	96
1	0110001	49	a	1100001	97
2	0110010	50	b	1100010	98
3	0110011	51	c	1100011	99
4	0110100	52	d	1100100	100
5	0110101	53	e	1100101	101
6	0110110	54	f	1100110	102
7	0110111	55	g	1100111	103
8	0111000	56	h	1101000	104
9	0111001	57	i	1101001	105
:	0111010	58	j	1101010	106
;	0111011	59	k	1101011	107
<	0111100	60	l	1101100	108
=	0111101	61	m	1101101	109
>	0111110	62	n	1101110	110
?	0111111	63	o	1101111	111
@	1000000	64	p	1110000	112
A	1000001	65	q	1110001	113
B	1000010	66	r	1110010	114
C	1000011	67	s	1110011	115
D	1000100	68	t	1110100	116
E	1000101	69	u	1110101	117
F	1000110	70	v	1110110	118
G	1000111	71	w	1110111	119
H	1001000	72	x	1111000	120
I	1001001	73	y	1111001	121
J	1001010	74	z	1111010	122
K	1001011	75	{	1111011	123
L	1001100	76	|	1111100	124
M	1001101	77	}	1111101	125
N	1001110	78	~	1111110	126
O	1001111	79	*Del*	1111111	127

Acest cod este 7-poziţional şi include $2^7 = 128$ de simboluri. Primele 32 de simboluri (cuvintele binare 0000000, 0000001, 0000010, …, 0011111) specifică detaliile tehnice ale transmisiunilor de informaţii şi nu au fost incluse în tabel.

Cuvintele binare 0100000, 0100001, 0100010, …, 1111110 reprezintă caracterele imprimabile din textele în limba engleză. Cuvântul 1111111 reprezintă caracterul neimprimabil *Delete* (Anulare).

Codificarea mesajelor se realizează prin înlocuirea simbolurilor cu cuvintele binare respective. De exemplu, cuvântul START se reprezintă în codul ASCII prin următoarea secvență de cuvinte binare:

1010011 1010100 1000001 1010010 1010100.

Evident, decodificarea se va realiza în ordine inversă. De exemplu, secvența de cuvinte binare

1010011 1010100 1001111 1010000

reprezintă în codul ASCII cuvântul STOP.

De regulă, limbajele de programare operează nu cu cuvintele binare propriu-zise, dar cu echivalentele lor zecimale. În programele PASCAL echivalentele zecimale ale caracterelor pot fi aflate cu ajutorul funcției predefinite `ord`. De exemplu:

```
ord('S')=83;   ord('T')=84;   ord('A')=65;   ord('R')=82
```

etc. Funcția predefinită `chr` returnează caracterul care corespunde echivalentului zecimal indicat. Astfel,

```
chr(83)='S';   chr(84)='T';   chr(65)='A';   chr(82)='R'.
```

Într-un mod similar, în programele C++ scriem:

```
int('S')=83;   int('T')=84;   int('A')=65;   int('R')=82
char(83)='S';  char(84)='T';  char(65)='A';  char(82)='R'.
```

Orientat la textele engleze, codul ASCII nu include literele cu semne diacritice şi caracterele grafice speciale întâlnite în diferite limbi europene şi în lucrările ştiinţifice. De aceea pentru calculatoarele moderne s-au elaborat versiuni dedicate ale codului ASCII, denumite **coduri ASCII extinse**. Codurile extinse sunt 8-poziționale şi includ $2^8 = 256$ de simboluri. Structura codurilor respective este prezentată în *tabelul 2.4*.

Tabelul 2.4

Structura codurilor ASCII extinse

Simbol	Cuvânt binar	Echivalent zecimal	Observații
Spațiu	00100000	32	Partea 1: – simbolurile codului ASCII
!	00100001	33	
"	00100010	34	
#	00100011	35	
…	…	…	
}	01111101	125	
~	01111110	126	
Del	01111111	127	

Continuarea tab. 2.4

Simbol	Cuvânt binar	Echivalent zecimal	Observații
А	10000000	128	Partea 2: – simboluri specifice limbilor naționale; – caractere grafice; – caractere ştiințifice
Б	10000001	129	
В	10000010	130	
…	…	…	
≡	11110000	240	
Ă	11110001	241	
ă	11110010	242	
Â	11110011	243	
â	11110100	244	
Î	11110101	245	
î	11110110	246	
Ş	11110111	247	
ş	11111000	248	
’	11111001	249	
⁻	11111010	250	
√	11111011	251	
Ţ	11111100	252	
ţ	11111101	253	
□	11111110	254	
	11111111	255	

Partea 1 a fiecărui cod extins include simbolurile de la 0 la 127 oferite de codul ASCII. ***Partea a 2-a*** este specifică fiecărei țări şi include simbolurile de la 128 la 255. Aceste simboluri sunt utilizate pentru reprezentarea literelor alfabetelor naționale, precum şi a caracterelor ştiințifice frecvent utilizate. Pentru exemplificare, în *tabelul 2.4* sunt prezentate codurile literelor *Ă, ă, Â, â, Î, î, Ş, ş, Ţ, ţ* din alfabetul limbii române propuse în anul 1992 de firma *TISH* (Chişinău). Este firesc ca utilizarea codurilor extinse să asigure prelucrarea informațiilor prezentate în diferite limbi.

Un alt exemplu de cod alfanumeric este codul binar 8-pozițional **EBCDIC** (*Extended Binary Coded Data Interchange Code*), care se utilizează pe calculatoarele mari.

Menționăm că extinderea domeniului de aplicare a codurilor 8-poziționale a favorizat utilizarea octetului şi a multiplilor lui pentru măsurarea cantității de informație:

1 *octet* = $2^3 = 8$ *biți*;

1 *Kilooctet* = $2^{10} \approx 10^3$ *octeți*;

1 *Megaoctet* = $2^{20} \approx 10^6$ *octeți*;

1 *Gigaoctet* = $2^{30} \approx 10^9$ *octeți*;

1 *Teraoctet* = $2^{40} \approx 10^{12}$ *octeți*;

1 *Petaoctet* = $2^{50} \approx 10^{15}$ *octeți*.

În literatura de specialitate octetul este notat prin *B* (*byte*), iar multiplii respectivi prin *KB, MB, GB, TB şi PB*.

Este cunoscut faptul că informatica este supusă într-o măsură foarte mare fenomenului globalizării, produsele-program şi echipamentele digitale fiind concepute în aşa mod, încât ele pot procesa informaţii prezentate în diverse limbi. Pentru reprezentarea pe calculator a caracterelor din aproape toate limbile globului, cel mai des se utilizează codul UNICODE, lungimea cuvintelor căruia poate ajunge până la 32 de cifre binare.

Întrebări şi exerciţii

❶ Câte mesaje posibile pot fi codificate cu ajutorul unui cod *m*-poziţional?

❷ Determinaţi codurile admise de sistemul de operare cu care lucraţi dvs.

❸ Codificaţi în codul *Gray* următoarele şiruri de cifre zecimale: 123, 461, 952, 783, 472.

❹ Decodificaţi mesajele reprezentate în codul *Aiken*:

a) `0011 1111 0100`
b) `1111 0000 0100`
c) `0010 0001 1011`
d) `1110 0010 1101`
e) `0011 1100 1111`
f) `1111 1101 0000`

❺ Codificaţi în codul ASCII expresiile:

a) `A+B`
b) `FOR I=1 TO N`
c) `PRINT A$`
d) `NEXT I`
e) `PAUSE`
f) `PROGRAM`

❻ Decodificaţi mesajele reprezentate în codul ASCII:

a) `1000010 1100101 1100111 1101001 1101110;`
b) `1010011 1110100 1101111 1110000;`
c) `1000101 1101110 1100100;`
d) `1101001 0111010 0111101 0110001 0111011.`

❼ Elaboraţi un program care afişează pe ecran codurile următoarelor caractere, introduse de la tastatură:
a) cifrele zecimale `0`, `1`, `2`, ..., `9`;
b) literele latine mari `A`, `B`, `C`, ..., `Z`;
c) literele latine mici `a`, `b`, `c`, ..., `z`;
d) semnele operaţiilor aritmetice;
e) caracterele speciale `;`, `<`, `=`, `>`, `?`, `[`, `]`, `{`, `}`, `/`, `\`.

❽ STUDIU DE CAZ. Cu ajutorul unui motor de căutare, găsiţi pe Internet descrierile codurilor frecvent utilizate. Stabiliţi parcursul istoric al codurilor respective, avantajele şi neajunsurile acestora, domeniile lor de utilizare.

2.4. Informaţia mesajelor continue

Sursele de informație studiate până în prezent se definesc printr-o variabilă S, care poate lua valori dintr-o mulțime finită de elemente distincte $\{s_1, s_2, ..., s_n\}$, denumită mulțimea mesajelor posibile. Practica ne demonstrează că nu toate sursele de informație pot fi definite direct în acest mod. Drept exemplu vom remarca termometrele cu mercur sau alcool, vitezometrele automobilelor, microfoanele, camerele de luat vederi etc. Astfel de surse pot fi definite printr-o variabilă S (temperatura, viteza momentană, tensiunea la bornele de ieşire ale microfonului etc.) care poate lua orice valoare într-un anumit interval $[s_{min}, s_{max}]$.

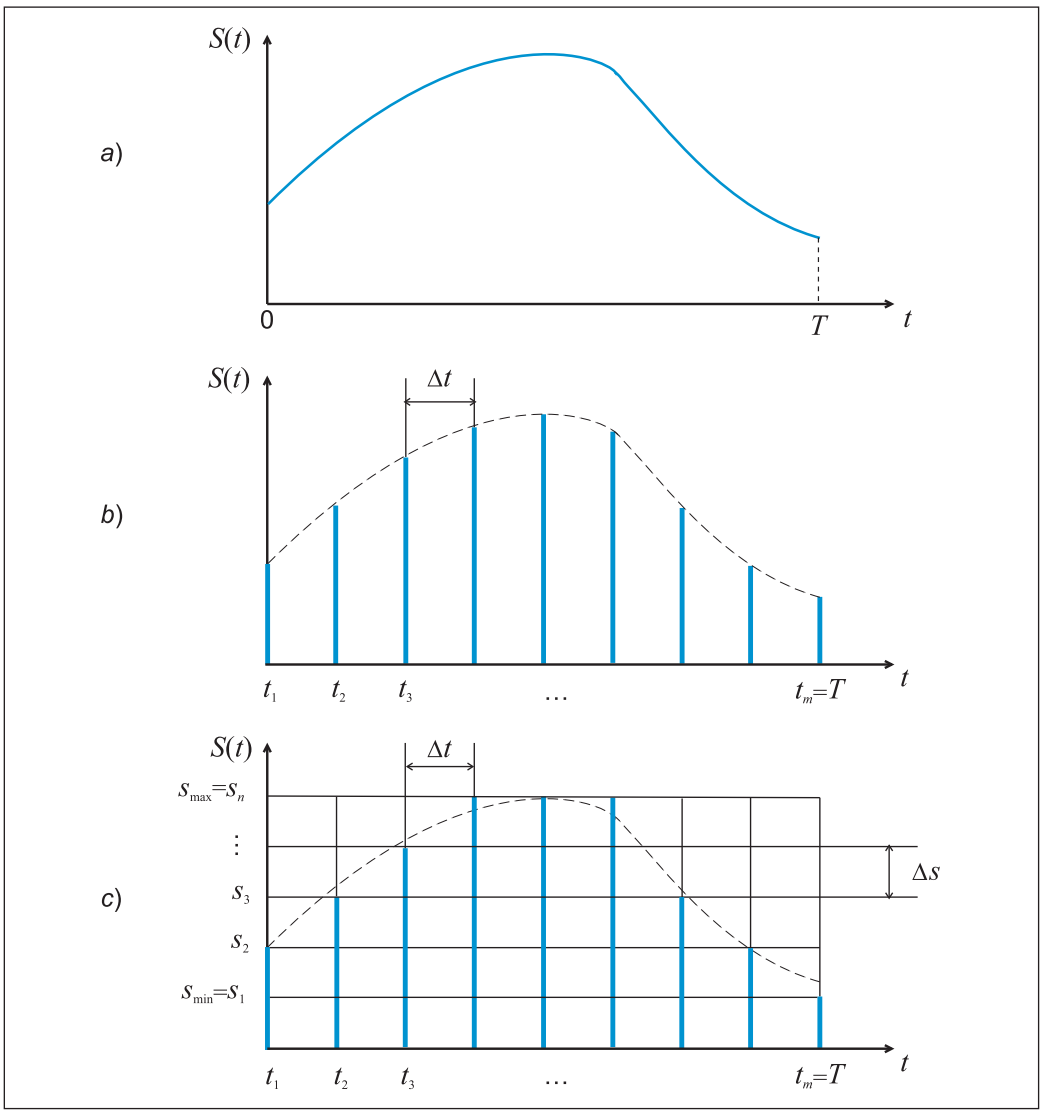


Fig. 2.4. Discretizarea mesajelor continue: *a* – mesaj continuu; *b* – mesaj discretizat în timp (eşantionat); *c* – mesaj eşantionat şi discretizat în valoare (cuantificat)

Sursele de informație care sunt definite printr-o variabilă S ce ia valori dintr-o mulțime finită de elemente distincte se numesc **surse cu mesaje discrete**. Sursele care se definesc printr-o variabilă S ce poate lua orice valori într-un anumit interval se numesc **surse cu mesaje continue**.

Ca şi în cazul surselor discrete, mesajele continue se produc în timp. Prin urmare, S este o funcție de timp, $S = S(t)$. În scopul evaluării cantității de informație în mesajele continue, vom analiza valorile funcției $S(t)$ numai în momentele de timp $t_1, t_2, ..., t_m$ (*fig. 2.4*). Valorile date, notate prin $S(t_1), S(t_2), ..., S(t_m)$, se numesc **eşantioane** (din franceză *echantillion* „cantitate mică luată dintr-un produs pentru a determina calitatea lui").

De obicei, momentele de timp $t_1, t_2, ..., t_m$ se definesc conform relației:

$$t_i = t_{i-1} + \Delta t.$$

Mărimea Δt se numeşte **perioada de eşantionare**. Valoarea concretă a perioadei de eşantionare Δt este determinată de viteza cu care $S(t)$ variază în timp.

De exemplu, în meteorologie schimbările de temperatură se produc pe parcurs de ore şi $\Delta t = 1\ h$, iar în tehnica prelucrării semnalelor sonore $\Delta t = 5 \cdot 10^{-5}\ s$.

Operația de transformare a mesajelor continue în eşantioane se numeşte discretizare în timp sau eşantionare.

Evident, până la recepția mesajului continuu, valorile concrete ale eşantioanelor $S(t_1), S(t_2), ..., S(t_m)$ sunt necunoscute destinatarului. Se cunoaşte numai intervalul $[s_{min}, s_{max}]$ în care poate lua valori arbitrare orice eşantion. În scopul evaluării cantității de informație dintr-un eşantion, vom rotunji valoarea $S(t_i)$ la una din valorile prestabilite $s_1, s_2, ..., s_n$ (*fig. 2.4*).

Valorile prestabilite $s_1, s_2, ..., s_n$ se numesc **cuante**, iar operația de transformare a valorilor curente ale mesajelor continue în cuante se numeşte **discretizare în valoare** sau **cuantificare**.

De obicei,

$$s_1 = s_{min}, \quad s_2 = s_{min} + \Delta s, \quad ..., \quad s_i = s_{i-1} + \Delta s, \quad ..., \quad s_n = s_{max}.$$

Mărimea Δs reprezintă **pasul** sau **intervalul de cuantificare**. Valoarea concretă a intervalului de cuantificare depinde de natura fizică a sursei de informație, precizia de măsurare, puterea de rezoluție a destinatarului etc.

De exemplu, pentru un termometru medical $s_{min} = 34^\circ$, $s_{max} = 42^\circ$ şi $\Delta s = 0{,}1^\circ$. În meteorologie $s_{min} = -60^\circ$, $s_{max} = +60^\circ$, $\Delta s = 1^\circ$. Pentru vitezometrul unui automobil $s_{min} = 0$, $s_{max} = 150$ km/h, $\Delta s = 5$ km/h.

Numărul eşantioanelor m şi numărul cuantelor n se determină din următoarele relații (*fig. 2.4 b* şi *c*):

$$m = \frac{T}{\Delta t} + 1; \quad n = \frac{|s_{max} - s_{min}|}{\Delta s} + 1,$$

unde T este durata mesajului continuu.

Întrucât cuantele $s_1, s_2, ..., s_n$ pot fi considerate mesaje discrete, cantitatea de informație într-un eşantion

$$I = \log_2 n = \log_2 \left(\frac{|s_{max} - s_{min}|}{\Delta s} + 1 \right),$$

iar cantitatea de informație într-un mesaj continuu:

$$V = mI = \left(\frac{T}{\Delta t} + 1\right)\log_2\left(\frac{|s_{\max} - s_{\min}|}{\Delta s} + 1\right).$$

De exemplu, pentru termometrul medical

$$I = \log_2\left(\frac{|42-34|}{0{,}1} + 1\right) \approx 6{,}34 \text{ } biți,$$

iar pentru vitezometrul automobilului

$$I = \log_2\left(\frac{|150-0|}{5} + 1\right) \approx 4{,}95 \text{ } biți.$$

Cantitatea de informație dintr-o înregistrare audio, pentru care $n = 256$, $\Delta t = 5 \cdot 10^{-5}$ *s* și $T = 45$ min. este

$$V = \frac{45 \cdot 60}{5 \cdot 10^{-5}} \log_2 256 = 4{,}32 \cdot 10^8 \text{ } biți = 432 \text{ } Mbiți.$$

Dacă precizia de măsurare și puterea de rezoluție a destinatarului cresc, perioada de eșantionare Δt și pasul de cuantificare Δs pot fi micșorate. În consecință, va crește și cantitatea de informație dintr-un mesaj continuu.

Informația mesajelor continue poate fi reprezentată printr-un set de cuvinte binare. Pentru aceasta, cuantele s_1, s_2, ..., s_n se codifică exact la fel ca și oricare alte mesaje discrete. De exemplu, indicațiile unui termometru medical ($I \approx 6{,}34$ *biți*) pot fi codificate cu un cod 7-pozițional. Cel mai frecvent se utilizează codurile numerice directe (*tab. 2.2*), cuvântul de cod reprezentând numărul cuantei respective. În unele aplicații se utilizează codul *Gray*, care este insensibil la perturbații.

Dispozitivul care transformă mesajul continuu aplicat la intrare într-o succesiune de cuvinte de cod se numește **convertor analog-numeric**. Operația inversă, și anume transformarea cuvintelor de cod aplicate la intrare în valorile cuantelor respective, se efectuează cu ajutorul **convertoarelor numeric-analogice**. Utilizarea convertoarelor este necesară în cazurile în care informația de procesat este reprezentată prin mesaje continue: controlul proceselor tehnologice, dirijarea obiectelor aflate în mișcare, monitorizarea parametrilor fiziologici în medicină, filtrarea și mixajul semnalelor audio etc.

Întrebări și exerciții

❶ Care este diferența dintre sursele cu mesaje discrete și sursele cu mesaje continue?

❷ Dați câteva exemple de surse cu mesaje continue. Concretizați intervalul în care ia valori variabila ce definește sursa.

❸ Explicați cum se efectuează operația de eșantionare. Cum se alege perioada de eșantionare?

❹ Explicați cum se efectuează operația de cuantificare. Cum se alege pasul de cuantificare?

❺ Cum influențează valorile perioadei de eşantionare şi ale pasului de cuantificare cantitatea de informație extrasă dintr-un mesaj continuu?

❻ Altimetrul cu impulsuri al unui avion poate măsura înălțimi de la 100 m până la 20 km. Eroarea de măsurare nu depăşeşte 1 m. Pentru a efectua o măsurare sunt necesare 10^{-3} *s*. Determinați cantitatea de informație furnizată de altimetru timp de 5 ore de zbor.

❼ Temperatura din interiorul unui reactor chimic se înregistrează pe o bandă de hârtie milimetrică. Pe axa absciselor se indică timpul (1 mm reprezintă o oră), iar pe axa ordonatelor – temperatura (1 mm reprezintă 10 °C). Câtă informație conține o înregistrare efectuată timp de 30 de zile, dacă temperatura poate varia de la 80 până la 1000 °C?

❽ Pentru înregistrarea sunetului se utilizează microfoane, tensiunea de ieşire a cărora variază de la 0 până la 100 μ*V*. Aparatul de înregistrare nu distinge tensiunile valorile cărora diferă cu mai puțin de 0,1 μ*V*. Pentru a asigura o reproducere fidelă, în fiecare secundă se iau 40 000 de eşantioane. Câtă informație va furniza microfonul dat timp de 3 ore?

❾ Care este destinația convertoarelor analog-numerice şi numeric-analogice?

❿ Elaborați un program care introduce de la tastatură valorile curente ale eşantioanelor şi afişează pe ecran codurile cuantelor corespunzătoare.

⓫ Elaborați un program care simulează funcționarea unui convertor numeric-analog.

⓬ CERCETEAZĂ! Utilizând un motor de căutare, găsiți pe Internet descrierea aparatelor de înregistrare digitală a sunetelor. Aflați perioada de eşantionare şi pasul de cuantificare utilizate în digitalizarea mesajelor sonore, realizată de fiecare dintre aparatele respective.

⓭ STUDIU DE CAZ. Cu ajutorul unui motor de căutare, găsiți pe Internet descrierile aparatelor de înregistrare digitală a sunetelor, propuse de unitățile de comerț. Comparați aceste aparate din perspectiva modului de digitalizare a mesajelor sonore. Aflați cum parametrii de discretizare influențează costurile aparatelor de înregistrare a sunetelor, care sunt avantajele, neajunsurile şi domeniile de utilizare a acestora.

2.5. Cuantizarea imaginilor

Imagine se numeşte reprezentarea unui obiect, executată pe o suprafață prin acțiunea directă a utilizatorului sau prin intermediul unui echipament. Cu titlul de exemplu amintim desenele, fotografiile, imaginile formate de diverse sisteme optice, optico-mecanice sau optico-electronice: microscopul, telescopul, aparatele cinematografice, televiziunea etc.

Pentru a evalua cantitatea de informație, imaginea este împărțită în **microzone**, numite de cele mai multe ori **puncte** sau **pixeli**. Descompunerea imaginii în puncte se realizează cu ajutorul unui **rastru** (de la cuvântul latin *raster* „greblă”). Rastrul reprezintă o suprafață plană, în general dreptunghiulară, pe care sunt trasate două seturi de linii paralele, perpendiculare între ele (*fig. 2.5*). Densitatea

liniilor şi, respectiv, densitatea punctelor caracterizează puterea de rezoluție a echipamentelor pentru reproducerea sau formarea imaginilor.

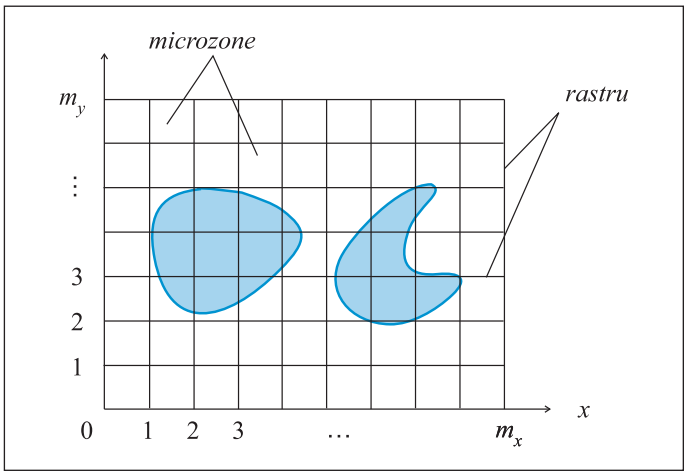


Fig. 2.5. Descompunerea imaginii în microzone

De exemplu, pentru ilustrațiile de gazetă se folosesc rastre cu rezoluția 24-30 linii/cm (576-900 de puncte pe 1 cm^2), iar pentru reproducerea tablourilor – rastre cu 54-60 linii/cm. Rastrul vizualizatorului, adică desenul pe care-l formează fasciculul de electroni pe ecranul tubului catodic, poate include 640 × 480, 800 × × 600, 720 × 400, ..., 1024 × 1024 de puncte.

Descompunerea imaginii în puncte (microzone) reprezintă o procedură de discretizare în spațiu.

În cazul imaginilor monocrome (alb-negru), fiecare microzonă se descrie prin **luminanța** (strălucirea) sa, care în general este o mărime continuă. Această mărime poate fi discretizată în valoare (cuantificată). Numărul cuantelor n va caracteriza puterea de rezoluție a echipamentelor pentru reproducerea sau formarea imaginilor. Prin urmare, cantitatea de informație a unei imagini monocrome:

$$I = m_x m_y \log_2 n,$$

unde m_x şi m_y reprezintă numărul de microzone ale rastrului respectiv pe orizontală şi verticală (*fig. 2.5*).

Întrucât culorile pot fi redate prin suprapunerea a trei reprezentări ale aceleiaşi imagini în roşu, verde şi albastru, cantitatea de informație dintr-o imagine color se determină din relația:

$$I = 3\, m_x m_y \log_2 n.$$

Imaginile obiectelor în mişcare se discretizează în timp, de obicei 24 (cinematograful) sau 25 (televizorul) de cadre pe secundă. Prin urmare, cantitatea de informație a unui film cu durata T se determină din relația:

$$V = T f I,$$

unde f este frecvența cadrelor, iar I cantitatea de informație dintr-un singur cadru.

De exemplu, în televiziune $m_x \approx m_y = 625$, $n = 32$ şi $f = 25$ de cadre pe secundă. Un cadru color va conține:

$$I = 3 \cdot 625 \cdot 625 \cdot \log_2 32 \approx 5{,}6 \text{ } \textit{Mbiți}.$$

Un film color cu durata de 1,5 ore va conține:

$$V = 1{,}5 \cdot 3\,600 \cdot 25 \cdot I \approx 791 \text{ } \textit{Gbiți}.$$

Setul de cuvinte binare care reprezintă informația microzonelor se numeşte imagine numerică. Operația de transformare a imaginii într-un set de cuvinte binare se numeşte cuantizarea imaginii.

Imaginile preluate de camerele video se cuantizează cu ajutorul convertoarelor analog-numerice. Imaginile de pe hârtie pot fi cuantizate cu ajutorul unui dispozitiv special, numit **scaner**. Acest dispozitiv conține celule fotosensibile, convertoare analog-numerice şi mecanisme de avansare a hârtiei.

Imaginile numerice se transformă în imagini propriu-zise cu ajutorul convertoarelor numeric-analogice şi al echipamentelor de formare a rastrului: tubul catodic şi sistemul de baleiere în vizualizatoare, matricea de ace în imprimantele mecanice etc.

Întrebări şi exerciţii

❶ Numiți operațiile necesare pentru a cuantiza imaginea.

❷ Care este destinația rastrului? Din care considerente se alege densitatea liniilor unui rastru?

❸ Cum se evaluează cantitatea de informație dintr-o imagine monocromă?

❹ Cum pot fi redate culorile unei imagini multicolore? Cum se evaluează cantitatea de informație dintr-o imagine color?

❺ Evaluați cantitatea de informație dintr-o fotografie de ziar cu dimensiunile 10×10 cm, redată cu ajutorul unui rastru ce conține 24 de puncte/cm. Fiecare punct poate avea următoarele nuanțe: alb, gri-deschis, gri-închis, negru.

❻ Câtă informație conține o fotografie color cu dimensiunile 20×20 cm, reprodusă cu ajutorul unui rastru ce conține 60 de puncte/cm? Pot fi redate până la 256 de niveluri de luminanță ale punctelor respective.

❼ Rastrul unei camere de luat vederi este format din 1024×1024 de puncte. Pot fi redate până la 64 de niveluri de luminanță ale punctelor respective. Câtă informație va conține un film video cu durata de 3 ore?

❽ Imaginea numerică conține câte un cuvânt binar pentru fiecare punct al rastrului unui vizualizator. Câte niveluri de luminanță pot fi redate pe ecran dacă cuvintele imaginii numerice sunt 3-poziționale? 5-poziționale? 8-poziționale?

❾ CERCETEAZĂ! Utilizând un motor de căutare, găsiți pe Internet descrierea aparatelor digitale fotografice. Aflați puterea de rezoluție realizată de fiecare dintre aparatele respective în scopul digitalizării imaginilor statice.

❿ STUDIU DE CAZ. Cu ajutorul unui motor de căutare, găsiți pe Internet descrierile aparatelor fotografice digitale, propuse de unitățile de comerț. Comparați aceste aparate din perspectiva modului de digitalizare a imaginilor statice.

Află cum parametrii de discretizare influențează costurile, avantajele, neajunsurile şi domeniile de utilizare a aparatelor fotografice digitale.

⓫ CERCETEAZĂ! Utilizând un motor de căutare, găsiți pe Internet descrierea camerelor video digitale. Aflați puterea de rezoluție realizată de fiecare dintre camerele respective în scopul digitalizării informațiilor dinamice.

⓬ STUDIU DE CAZ. Cu ajutorul unui motor de căutare, găsiți pe Internet descrierile camerelor video digitale propuse de unitățile de comerț. Comparați aceste camere din perspectiva modului de digitalizare a imaginilor dinamice. Aflați cum parametrii de discretizare influențează costurile camerelor video digitale, care sunt avantajele, neajunsurile şi domeniile de utilizare a acestora.

2.6. Reprezentarea şi transmiterea informaţiei

Obiectul material folosit pentru păstrarea, transmiterea sau prelucrarea informației se numeşte **purtător de informație**. Deosebim purtători statici şi purtători dinamici de informație.

Purtătorii statici se utilizează pentru păstrarea informației sau, cu alte cuvinte, pentru transmiterea ei în timp. Informația înregistrată pe un purtător static poate fi citită în scopul prelucrării sau utilizării ulterioare. Primii purtători statici folosiți de omenire au fost pietrele, plăcile de lut ars, papirusul. Un alt purtător static de informație îl constituie hârtia. Mesajele înregistrate pe hârtie sub formă de manuscrise, desene sau texte tipărite pot fi păstrate un timp foarte îndelungat. În calculatoare, ca purtători statici, se utilizează:

– hârtia pentru imprimantele mecanice, cu jet de cerneală, laser etc.;
– straturile active ale benzilor, ale discurilor magnetice;
– mediile reflectoare ale discurilor optice etc.

Transmiterea informației în spațiu se realizează cu ajutorul **purtătorilor dinamici**. În calitate de purtători dinamici tehnica actuală foloseşte:

– unde acustice în gaze (aer) sau lichide;
– tensiuni şi curenți electrici;
– unde electromagnetice etc.

Întrucât transmiterea informației se produce în spațiu şi timp, cel puțin o mărime fizică a purtătorului de informație trebuie să se schimbe.

> ***Se numeşte semnal variația mărimii fizice ce asigură transmiterea mesajelor. Caracteristica semnalului folosită pentru reprezentarea (descrierea) mesajelor se numeşte parametru informațional al semnalului.***

De exemplu, în transmisiunile radio şi TV, în calitate de purtători se utilizează undele electromagnetice. Amplitudinea sau frecvența acestor unde poate varia în timp (*fig. 2.6*, p. 110). În primul caz, parametrul informațional îl constituie amplitudinea oscilațiilor, iar în al doilea – frecvența lor.

În calculatoarele electronice în calitate de purtător de informație se foloseşte, de obicei, curentul electric, tensiunea şi intensitatea curentului fiind parametrii

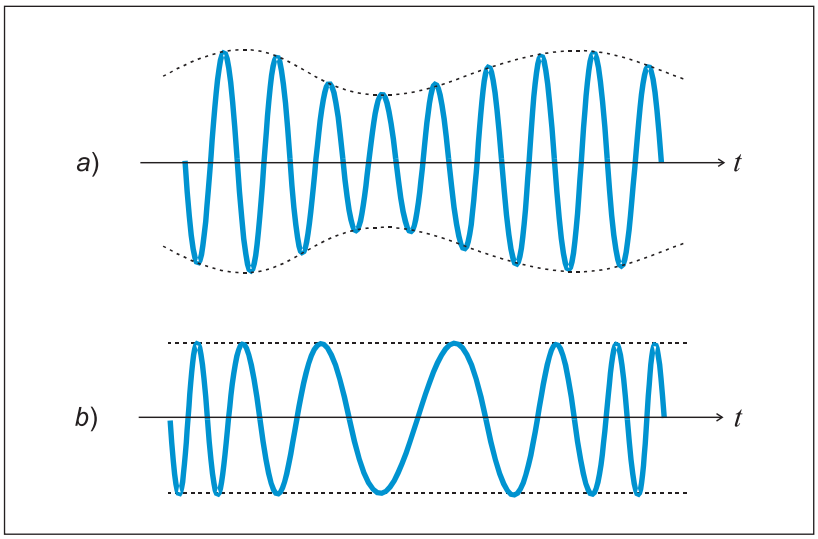


Fig. 2.6. Parametrii informaţionali ai undelor electromagnetice:
a – amplitudinea; *b* – frecvenţa

informaționali ai semnalului. Semnalele respective au formele prezentate în *figura 2.7*. În cazul nivelurilor de tensiune, unei valori a tensiunii i se asociază cifra binară 0, iar alteia cifra binară 1. Cifrele binare 0 şi 1 pot fi de asemenea asociate, respectiv, cu absență sau prezență de impuls sau cu impuls negativ şi impuls pozitiv.

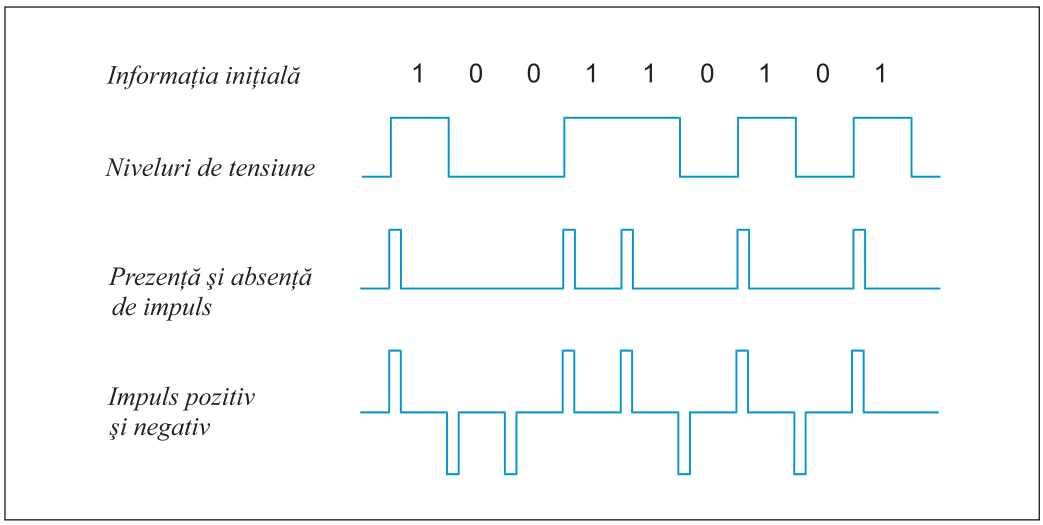


Fig. 2.7. Semnale utilizate în tehnica de calcul

Semnalul se numeşte discret, dacă parametrul informaţional respectiv poate lua numai un număr finit de valori. Semnalul se numeşte continuu, dacă parametrul informaţional poate lua orice valoare într-un anumit interval.

De exemplu, semnalele din *figura 2.6* sunt semnale continue, iar semnalele din *figura 2.7* sunt semnale discrete. Evident, semnalele discrete şi continue sunt forme de reprezentare a mesajelor respective.

Orice sistem tehnic utilizează acele semnale care-i asigură o realizare cât mai bună a funcţiilor pentru care a fost conceput. Calculatoarele actuale utilizează niveluri de tensiune, reţelele telefonice – curenţi electrici, iar radioul şi televiziunea – unde electromagnetice etc. Practica demonstrează că semnalele continue pot fi transmise la distanţe mult mai mari decât cele discrete. Prin urmare, pentru a asigura legătura dintre calculatoarele aflate la distanţă, la emisie semnalele discrete vor fi transformate în semnale continue. La recepţie se va realiza transformarea inversă: semnalul continuu va fi transformat într-un semnal discret.

Operaţia prin care parametrul informaţional al semnalului continuu se modifică în funcţie de valorile semnalului discret se numeşte modulare.

Dispozitivul tehnic care realizează operaţia în cauză se numeşte **modulator**. Pentru exemplificare, în *figura 2.8* sunt prezentate operaţiile de modulare în amplitudine şi frecvenţă.

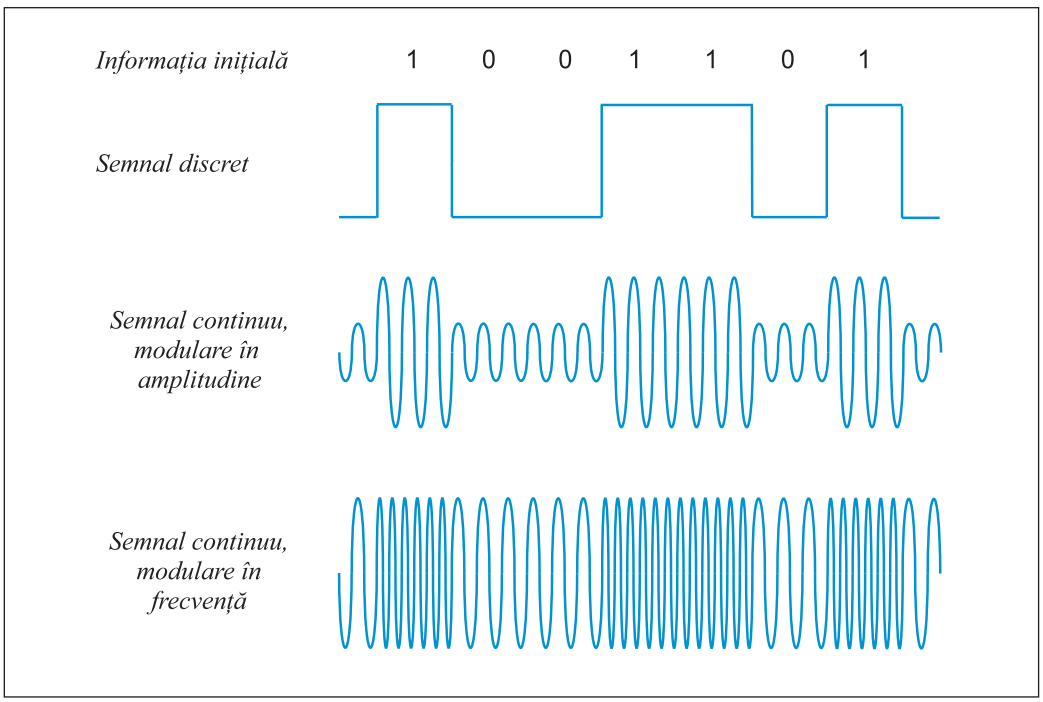


Fig. 2.8. Modularea semnalelor continue

Operaţia de extragere a semnalului discret dintr-un semnal continuu în funcţie de procedeul de modulare adoptat se numeşte demodulare.

Dispozitivul tehnic care realizează operaţia respectivă se numeşte **demodulator**.

Schema detaliată a sistemului de transmisie a informației este prezentată în *figura 2.9.*

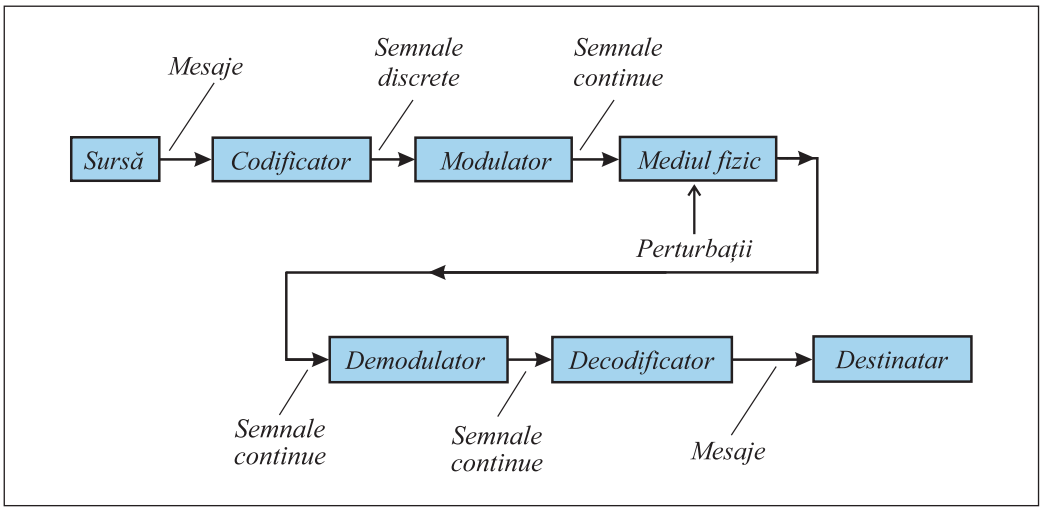


Fig. 2.9. Schema detaliată a unui sistem de transmisie a informaţiei

Amintim destinația componentelor sistemului examinat:

codificatorul – transformă mesajele emise de sursă în cuvinte binare;

modulatorul – transformă semnalele discrete ce reprezintă cuvintele binare în semnale continue;

mediul fizic – reprezintă conductorii, fibrele optice, eterul etc. prin care se propagă semnalele continue;

demodulatorul – transformă semnalele continue în semnale discrete;

decodificatorul – transformă cuvintele binare în mesaje.

Evident, în cazul utilizării codurilor corectoare şi a codurilor detectoare de erori, sistemul de transmisie va fi insensibil la perturbații.

Caracteristica principală a oricărui sistem de transmisie a informației este **capacitatea de transmisie**, exprimată în ***biți pe secundă***. Capacitatea de transmisie depinde de caracteristicile fizice ale componentelor sistemului, metodele de modulare-demodulare, caracteristicile statistice ale perturbațiilor. De exemplu, capacitatea de transmisie a unui canal telefonic este de circa 34 *Kbit*/s; capacitatea unui canal radio, cu unde centimetrice – de circa 1 *Gbit*/s; capacitatea unui canal optic – de 1 *Tbit*/s.

Întrebări şi exerciţii

❶ Care este deosebirea dintre purtătorii statici şi purtătorii dinamici de informație?

❷ Determinați tipul următorilor purtători de informație:

a) cartele perforate;
b) unde ultrasonore;
c) benzi perforate;
d) unde acustice;
e) pelicule fotosensibile;
f) unde gravitaționale;
g) hârtie fotografică.

❸ Numiți parametrii informaționali ai semnalelor emise de următoarele surse:
a) microfonul;
b) stația de radio, unde lungi, medii sau scurte;
c) instrumentul muzical;
d) stația de radio, unde ultrascurte;
e) camera de luat vederi.

❹ Descrieți purtătorii de informație și semnalele utilizate în calculatoarele actuale.

❺ Explicați operațiile de modulare și demodulare a semnalelor.

❻ Care este destinația modulatorului? Dar a demodulatorului?

❼ În *figura 2.8* sunt redate semnalele continue ce reprezintă cuvântul binar 1001101. În codul ASCII (*tab. 2.3*) acestui cuvânt îi corespunde simbolul `M`. Redați pe un desen semnalele continue ce corespund următoarelor simboluri:

a) `>`;
b) `r`;
c) `W`;
d) `9`;
e) `<`;
f) `K`;
g) `a`;
h) `@`.

❽ De ce depinde capacitatea de transmisie a canalului? În ce unități se măsoară ea?

❾ Cum influențează perturbațiile asupra capacității de transmisie a canalului?

❿ CERCETEAZĂ! Aflați capacitățile de transmisie a canalelor din componența sistemelor informatice la care ai acces la domiciliu, liceu, în locurile publice.

Capitolul 3

BAZELE ARITMETICE ALE TEHNICII DE CALCUL

3.1. Sisteme de numeraţie

În calculatoarele digitale informaţia de orice categorie este reprezentată, stocată şi prelucrată în formă numerică. Numerele se reprezintă prin simboluri elementare denumite **cifre**.

> ***Totalitatea regulilor de reprezentare a numerelor, împreună cu mulţimea cifrelor poartă denumirea de sistem de numeraţie. Numărul cifrelor defineşte baza sistemului de numeraţie.***

Prezentăm câteva exemple de sisteme de numeraţie:

– **sistemul zecimal** este un sistem de numeraţie în baza 10, numărul de cifre utilizate fiind 10, respectiv, 0, 1, 2, ..., 9;

– **sistemul binar** este un sistem de numeraţie în baza 2, numărul de cifre utilizate este 2, adică 0 şi 1. Cifrele respective se numesc **cifre binare** sau **biţi**;

– **sistemul ternar** este un sistem de numeraţie în baza 3, numărul de cifre utilizate fiind 3, respectiv, 0, 1 şi 2;

– **sistemul octal** este un sistem de numeraţie în baza 8, conţinând 8 cifre: 0, 1, 2, ..., 7;

– **sistemul hexazecimal** este un sistem de numeraţie în baza 16 şi conţine 16 cifre: 0, 1, 2, ..., 9, *A* (zece), *B* (unsprezece), *C* (doisprezece), *D* (treisprezece), *E* (paisprezece), *F* (cincisprezece).

În *tabelul 3.1* sunt date reprezentările unora şi aceloraşi numere în diferite baze.

Tabelul 3.1

Reprezentarea unor numere în diferite baze

Zecimal	Binar	Octal	Hexa-zecimal	Zecimal	Binar	Octal	Hexa-zecimal
0	0	0	0	7	111	7	7
1	1	1	1	8	1000	10	8
2	10	2	2	9	1001	11	9
3	11	3	3	10	1010	12	A
4	100	4	4	11	1011	13	B
5	101	5	5	12	1100	14	C
6	110	6	6	13	1101	15	D

Zecimal	Binar	Octal	Hexa-zecimal
14	1110	16	E
15	1111	17	F
16	10000	20	10
17	10001	21	11
18	10010	22	12
19	10011	23	13
20	10100	24	14
. . .	. . .	. . .	. . .
30	11110	36	1E
. . .	. . .	. . .	. . .
40	101000	50	28
. . .	. . .	. . .	. . .

Zecimal	Binar	Octal	Hexa-zecimal
50	110010	62	32
. . .	. . .	. . .	. . .
60	111100	74	3C
. . .	. . .	. . .	. . .
70	1000110	106	46
. . .	. . .	. . .	. . .
80	1010000	120	50
. . .	. . .	. . .	. . .
90	1011010	132	5A
. . .	. . .	. . .	. . .
100	1100100	144	64
. . .	. . .	. . .	. . .

Regula de reprezentare a numerelor în sistemul zecimal rezultă din următorul exemplu:

$$(3856{,}43)_{10}=3\cdot10^{3}+8\cdot10^{2}+5\cdot10^{1}+6\cdot10^{0}+4\cdot10^{-1}+3\cdot10^{-2}.$$

Se observă că în această reprezentare semnificația (valoarea) fiecărei cifre depinde de poziția pe care o ocupă în număr. De exemplu, cifra 3 se întâlnește de 2 ori: prima dată are semnificația 3 000, iar a doua oară – semnificația 0,03.

Sistemele în care semnificația cifrelor depinde de poziția ocupată în cadrul numerelor se numesc sisteme de numerație poziționale.

Presupunem că numărul N are partea întreagă formată din $n+1$ cifre, iar partea fracționară – din m cifre:

$$N=c_nc_{n-1}...c_1c_0,c_{-1}c_{-2}...c_{-m}.$$

Valoarea acestui număr se evaluează în funcție de baza sistemului:

$$(N)_b=c_nb^n+c_{n-1}b^{n-1}+...+c_1b^1+c_0b^0+c_{-1}b^{-1}+c_{-2}b^{-2}+...+c_{-m}b^{-m}.$$

Efectuând calculele respective, se va realiza **conversiunea** numărului $(N)_b$ din baza b în sistemul zecimal.

De exemplu,

$(101,1)_{10} = 1\cdot10^{2} + 0\cdot10^{1} +1\cdot10^{0} + 1\cdot10^{-1} = 101{,}1;$

$(101,1)_{2} = 1\cdot2^{2} + 0\cdot2^{1} +1\cdot2^{0} + 1\cdot2^{-1} = 5{,}5;$

$(101,1)_{3} = 1\cdot3^{2} + 0\cdot3^{1} +1\cdot3^{0} + 1\cdot3^{-1} = 10{,}333...;$

$(101,1)_{8} = 1\cdot8^{2} + 0\cdot8^{1} +1\cdot8^{0} + 1\cdot8^{-1} = 65{,}125;$

$(101,1)_{16} = 1\cdot16^{2} + 0\cdot16^{1} +1\cdot16^{0} + 1\cdot16^{-1} = 257{,}0625.$

Formal, sistemul zecimal nu prezintă niciun avantaj deosebit față de alte sisteme de numerație. Se presupune că acest sistem a fost adoptat încă din cele mai vechi timpuri datorită faptului că procesul de numărare a folosit ca instrumente inițiale degetele mâinilor.

Un calculator poate fi prevăzut să funcționeze în orice sistem de numerație. Pe parcursul dezvoltării tehnicii de calcul, s-a stabilit că cel mai avantajos este sistemul binar. Acest sistem a fost preferat din următoarele motive:

– simplitatea regulilor pentru operațiile aritmetice şi logice;

– materializarea fizică a cifrelor în procesul prelucrării sau stocării numerelor se face mai uşor pentru două simboluri decât pentru zece: perforat–neperforat, contact închis–contact deschis, prezență sau absență de curent etc.;

– circuitele care trebuie să diferențieze numai între două stări sunt mai sigure în funcționare decât cele care trebuie să diferențieze între zece stări.

Menționăm că în procesul dezvoltării civilizației umane au fost create şi **sisteme de numerație nepoziționale**. Drept exemplu poate servi **sistemul roman**, care utilizează cifrele *I, V, X, L, C, D, M*. Întrucât regulile de reprezentare a numerelor şi de efectuare a operațiilor aritmetice sunt foarte complicate, sistemele nepoziționale au o utilizare foarte restrânsă.

Întrebări şi exerciţii

❶ Cum se defineşte un sistem de numerație?

❷ Care este deosebirea dintre sistemele de numerație poziționale şi nepoziționale?

❸ Dați exemple de sisteme de numerație poziționale. Cum se defineşte baza sistemului de numerație?

❹ Evaluați numărul $(101,1)_b$ scris în următoarele sisteme de numerație:

$b = 4, 5, 6, 7, 9, 11, 12, 13, 14$ şi 15.

❺ Evaluați numerele ce urmează:

a)	$(328)_9$;	*i)*	$(1010,01)_3$;	*q)*	$(341,02)_8$;
b)	$(516)_7$;	*j)*	$(201,12)_4$;	*r)*	$(ABCD)_{16}$;
c)	$(1010,01)_2$;	*k)*	$(341,02)_6$;	*s)*	$(328)_{16}$;
d)	$(201,12)_3$;	*l)*	$(1111)_{16}$;	*t)*	$(516)_{16}$;
e)	$(341,02)_5$;	*m)*	$(328)_{11}$;	*u)*	$(1010,01)_{16}$;
f)	$(FFFF)_{16}$;	*n)*	$(516)_9$;	*v)*	$(201,12)_{16}$;
g)	$(328)_{10}$;	*o)*	$(1010,01)_8$;	*w)*	$(341,02)_{12}$;
h)	$(516)_8$;	*p)*	$(201,12)_8$;	*x)*	$(F001)_{16}$.

❻ Care factori au contribuit la utilizarea în tehnica de calcul a sistemului binar?

❼ Elaborați un program care evaluează numerele scrise în baza b, $b \le 10$.

❽ Elaborați un program care evaluează numerele scrise în baza b, $10 \le b \le 36$.

❾ CERCETEAZĂ! Explorând spațiul digital virtual, aflați ce sisteme de numerație utilizau marile civilizații antice. Cum au influențat aceste sisteme de numerație civilizațiile moderne?

3.2. Conversiunea numerelor dintr-un sistem în altul

Conversiunea numărului $(N)_b$ în echivalentul său zecimal se efectuează conform formulei din paragraful precedent:

$$(N)_b=c_nb^n+c_{n-1}b^{n-1}+...+c_1b^1+c_0b^0+c_{-1}b^{-1}+c_{-2}b^{-2}+...+c_{-m}b^{-m}.$$

Conversiunea numărului zecimal $(N)_{10}$ în echivalentul său în baza b se efectuează conform următoarelor reguli:

– se împarte la baza respectivă partea întreagă şi câturile obținute după fiecare împărțire, până se obține câtul zero; rezultatul conversiunii părții întregi este constituit din resturile obținute, considerate în ordinea inversă de apariție;

– se înmulțește cu baza partea fracționară, apoi toate părțile fracționare obținute din produsul anterior, până când partea fracționară a unui produs este zero sau până la obținerea unui număr de cifre fracționare dorit; rezultatul conversiunii părții fracționare este constituit din părțile întregi ale produselor, considerate în ordinea apariției.

Să analizăm câteva exemple.

1) Să se transforme numărul zecimal 53,40625 în echivalentul său binar.

$53 : 2 = 26 + {}^1/_2$;
$26 : 2 = 13 + {}^0/_2$;
$13 : 2 = 6 + {}^1/_2$;
$6 : 2 = 3 + {}^0/_2$;
$3 : 2 = 1 + {}^1/_2$;
$1 : 2 = 0 + {}^1/_2$.

Prin urmare, partea întreagă a numărului binar va fi 110101.

$0{,}40625 \times 2 = 0{,}8125$;
$0{,}8125 \times 2 = 1{,}625$;
$0{,}625 \times 2 = 1{,}25$;
$0{,}25 \times 2 = 0{,}5$;
$0{,}5 \times 2 = 1{,}0$.

Partea fracționară a numărului binar va fi 01101. Prin urmare,

$$(53{,}40625)_{10}=(110101{,}01101)_2.$$

2) Să se transforme numărul 23,7 din sistemul zecimal în sistemul binar.

$23 : 2 = 11 + {}^1/_2$;
$11 : 2 = 5 + {}^1/_2$;
$5 : 2 = 2 + {}^1/_2$;

$2 : 2 = 1 + {}^{0}/_{2}$;
$1 : 2 = 0 + {}^{1}/_{2}$;

$0{,}7 \times 2 = 1{,}4$;
$0{,}4 \times 2 = 0{,}8$;
$0{,}8 \times 2 = 1{,}6$;
$0{,}6 \times 2 = 1{,}2$;
$0{,}2 \times 2 = 0{,}4$;
$0{,}4 \times 2 = 0{,}8$;
...

Se observă că operația poate fi continuată la infinit, adică nu există o conversiune exactă a numărului analizat. Prin urmare,

$$(23{,}7)_{10}=(10111{,}101100...)_2.$$

3) Să se efectueze conversiunea numărului 1996,0625 din sistemul zecimal în sistemul octal.

$1996 : 8 = 249 + {}^{4}/_{8}$;
$249 : 8 = 31 + {}^{1}/_{8}$;
$31 : 8 = 3 + {}^{7}/_{8}$;
$3 : 8 = 0 + {}^{3}/_{8}$;

$0{,}0625 \times 8 = 0{,}5$;
$0{,}5 \times 8 = 4$.

Prin urmare,

$$(1996{,}0625)_{10}=(3714{,}04)_8.$$

4) Să se transforme numărul 2914,25 din sistemul zecimal în sistemul hexazecimal.

$2914 : 16 = 182 + {}^{2}/_{16}$;
$182 : 16 = 11 + {}^{6}/_{16}$;
$11 : 16 = 0 + {}^{11}/_{16}$;

$0{,}25 \times 16 = 4$.

Prin urmare,

$$(2914{,}25)_{10} = (B62{,}4)_{16}.$$

Întrebări şi exerciţii

❶ Cum se efectuează conversiunea unui număr dintr-un sistem cu baza b în sistemul zecimal?

❷ Transformați în sistemul zecimal numerele ce urmează:

a) $(100001{,}01111)_2$; *b)* $(328{,}678)_9$; *c)* $(100{,}100)_{16}$;

d) $(10,01)_{16}$; *g)* $(1221,1112)_3$; *j)* $(4231,124)_5$;

e) $(AAA,BBB)_{16}$; *h)* $(1321,1312)_4$; *k)* $(50,505050)_6$;

f) $(EE,00F)_{16}$; *i)* $(124,521)_7$; *l)* $(7777,001)_8$.

❸ Cum se efectuează conversiunea unui număr zecimal în echivalentul său în baza b?

❹ Transformați în sistemul binar numerele zecimale ce urmează:

a) 13,889; *e)* 93,447; *i)* 58,749;

b) 38,668; *f)* 70,212; *j)* 4,345;

c) 53,536; *g)* 8,347; *k)* 3,156;

d) 29,261; *h)* 39,764; *l)* 91,428.

Verificați rezultatele efectuând conversiunea binar-zecimală.

❺ Transformați în sistemul octal numerele zecimale ce urmează:

a) 358,932; *e)* 886,526; *i)* 795,128;

b) 479,093; *f)* 971,258; *j)* 680,895;

c) 591,241; *g)* 515,914; *k)* 256,453;

d) 649,113; *h)* 347,607; *l)* 838,261.

Verificați rezultatele efectuând conversiunea octal-zecimală.

❻ Transformați în sistemul hexazecimal numerele zecimale ce urmează:

a) 1424,699; *e)* 5818,961; *i)* 4985,995;

b) 3517,315; *f)* 9336,491; *j)* 9721,678;

c) 9607,201; *g)* 3442,722; *k)* 5292,837;

d) 8974,664; *h)* 4521,449; *l)* 2734,592.

Verificați rezultatele efectuând conversiunea hexazecimal-zecimală.

❼ Elaborați un program care transformă numerele zecimale în echivalentele respective din sistemul de numerație în baza b, $b<10$.

❽ Elaborați un program pentru transformarea numerelor zecimale în numere din sistemul în baza b, $10<b\leq 36$.

3.3. Conversiunea din binar în octal, hexazecimal și invers

Întrucât $8 = 2^3$, **conversiunea binar-octală** și **octal-binară** se poate face direct. Orice cifră octală se reprezintă prin 3 cifre binare:

0 = 000; 1 = 001;

2 = 010;	5 = 101;
3 = 011;	6 = 110;
4 = 100;	7 = 111.

Dacă este dat un număr octal, pentru conversiunea lui în binar se va scrie fiecare cifră octală prin 3 cifre binare.

Exemple:

$(247,315)_8 = (010\ 100\ 111,\ 011\ 001\ 101)_2$;

$(512,07)_8 = (101\ 001\ 010,\ 000\ 111)_2$;

$(3,146)_8 = (011,\ 001\ 100\ 110)_2$.

Dacă este considerat un număr binar, pentru conversiunea lui în octal se vor grupa câte 3 cifre binare pornind de la poziția virgulei spre stânga pentru partea întreagă, respectiv dreapta pentru partea fracționară, aflând corespondentul în octal. Pentru completarea unui grup de trei cifre binare, zerourile din fața numărului, respectiv după ultima cifră a părții fracționare, nu modifică semnificația numărului.

Exemple:

$(11,011101)_2 = (011,011\ 101)_2 = (3,35)_8$;

$(10,11011)_2 = (010,110\ 110)_2 = (2,66)_8$;

$(1001,01011)_2 = (001\ 001,010\ 110)_2 = (11,26)_8$.

În mod similar se procedează şi în cazul sistemului hexazecimal, baza căruia este $16 = 2^4$. Orice cifră hexazecimală se reprezintă prin 4 cifre binare:

0 = 0000;	8 = 1000;
1 = 0001;	9 = 1001;
2 = 0010;	A = 1010;
3 = 0011;	B = 1011;
4 = 0100;	C = 1100;
5 = 0101;	D = 1101;
6 = 0110;	E = 1110;
7 = 0111;	F = 1111.

Exemple:

$(6AF3,B2)_{16} = (0110\ 1010\ 1111\ 0011,\ 1011\ 0010)_2$;

$(6F1,3CA)_{16} = (0110\ 1111\ 0001,\ 0011\ 1100\ 1010)_2$;

$(11,01101)_2 = (0011,\ 0110\ 1000)_2 = (3,68)_{16}$;

$(10001,01011)_2 = (0001\ 0001,\ 0101\ 1000)_2 = (11,58)_{16}$.

Întrebări şi exerciţii

❶ Cum se efectuează conversiunea octal-binară şi binar-octală?

❷ Transformați în sistemul binar numerele octale ce urmează:

a)	15,006;	*e)*	21,626;	*i)*	42,322;
b)	13,06;	*f)*	6,3415;	*j)*	44,523;
c)	43,15;	*g)*	771,25;	*k)*	32,271;
d)	10,01;	*h)*	12,121;	*l)*	73,536.

❸ Transformați în sistemul octal numerele binare ce urmează:

a)	1,1;	*e)*	1101,1;	*i)*	10110,001011;
b)	101,10101;	*f)*	1,000001;	*j)*	11111,0010001;
c)	1111,000101;	*g)*	11110001,101;	*k)*	11001,00101;
d)	10101110,1001;	*h)*	0,00001;	*l)*	1011,0001011.

❹ Elaborați un program pentru conversiunea binar-octală şi octal-binară.

❺ Cum se efectuează conversiunea hexazecimal-binară şi binar-hexazecimală?

❻ Transformați în sistemul binar numerele hexazecimale ce urmează:

a)	FFF,AAA;	*e)*	3,1AB;	*i)*	C,DC1;
b)	F,1A;	*f)*	AABB,0000F;	*j)*	942,14A;
c)	1,009;	*g)*	81,91A;	*k)*	CAA,B;
d)	0,00F;	*h)*	10,01;	*l)*	DAD,ABA.

❼ Transformați în sistemul hexazecimal numerele binare ce urmează:

a)	1001011101,01101;	*e)*	1011101,011;	*i)*	1011,0101;
b)	111,01;	*f)*	11,001011101;	*j)*	11001,0110;
c)	1,1;	*g)*	11110,01111;	*k)*	0,00001;
d)	10101110111,00101;	*h)*	111011,0010000;	*l)*	101,01011.

❽ Elaborați un program pentru conversiunea binar-hexazecimală şi hexazecimal-binară.

❾ Transformați în sistemul hexazecimal numerele octale ce urmează:

a)	13,146;	*e)*	451,35;	*i)*	644,031;
b)	613,12;	*f)*	12,357;	*j)*	5,4312;
c)	541,723;	*g)*	53,627;	*k)*	675,542;
d)	104,527;	*h)*	572,004;	*l)*	372,271.

❿ Transformați în sistemul octal numerele hexazecimale ce urmează:

a)	AA;	*e)*	15F,6A1;	*i)*	BBB;
b)	A2B,1F;	*f)*	3,281;	*j)*	AF,31B;
c)	F,3A;	*g)*	9,AF;	*k)*	2C,ACB;
d)	FFFF;	*h)*	3,418F;	*l)*	A1B,39E.

⓫ Elaborați un program pentru conversiunea octal-hexazecimală și hexazecimal-octală.

3.4. Operații aritmetice în binar

Operațiile aritmetice cu numerele binare sunt foarte simple. Regulile de operare în sistemul binar sunt prezentate în *tabelele 3.2, 3.3* și *3.4*.

Tabelul 3.2
Adunarea binară

0 + 0 = 0
0 + 1 = 1
1 + 0 = 1
1 + 1 = 10

Tabelul 3.3
Scăderea binară

0 – 0 = 0
1 – 0 = 1
1 – 1 = 0
10 – 1 = 1

Tabelul 3.4
Înmulțirea binară

$0 \times 0 = 0$
$0 \times 1 = 0$
$1 \times 0 = 0$
$1 \times 1 = 1$

Exemple:

1) Se propune adunarea în binar a numerelor zecimale 29 și 43:
$(29)_{10} = (11101)_2$;
$(43)_{10} = (101011)_2$;

11101 +
101011
1001000

Verificare: $(1001000)_2 = (72)_{10}$, rezultatul este corect, întrucât $(29)_{10} + (43)_{10} = (72)_{10}$.

2) Se propune scăderea în binar a numărului zecimal 37 din numărul zecimal 46:
$(37)_{10} = (100101)_2$;
$(46)_{10} = (101110)_2$;

101110_
100101
1001

Verificare: $(1001)_2 = (9)_{10}$, rezultatul este corect, întrucât $(46)_{10} – (37)_{10} = (9)_{10}$.

3) Se propune înmulțirea în binar a numerelor zecimale 3,25 și 7,125:
$(3,25)_{10} = (11,01)_2$;
$(7,125)_{10} = (111,001)_2$;

11,01	×
111,001	
1101	
0000	
0000	
1101	
1101	
1101	
10111,00101	

Verificare: $(10111,00101)_2 = (23,15625)_{10}$, rezultatul este corect, întrucât $(3,25)_{10} \times$ $\times (7,125)_{10} = (23,15625)_{10}$.

4) Se propune împărțirea în binar a numărului zecimal 211 la numărul zecimal 3:

$(211)_{10} = (11010011)_2$;

$(3)_{10} = (11)_2$;

11010011	11
11	1000110
00100	
11	
11	
11	
01	

Deci 11010011 : 11=1000110, rest 1.

Verificare: $(1000110)_2 = (70)_{10}$, rezultatul este corect, întrucât $(211)_{10} : (3)_{10} =$ $= (70)_{10} + (1)_{10}$.

Menționăm că atât la înmulțire, cât şi la împărțire, fixarea virgulei care desparte partea întreagă de cea fracționară se face la fel ca şi în sistemul de numerație zecimal.

Întrebări şi exerciţii

❶ Cum se execută operațiile aritmetice în sistemul binar?

❷ Calculați în sistemul binar:

a)	34 + 251;	*f)*	1996 – 51;	*k)*	0,5 × 0,5;
b)	68 – 7;	*g)*	2015 + 1995;	*l)*	1 : 0,5;
c)	1512 + 620;	*h)*	28,5 + 0,75;	*m)*	40 : 0,125;
d)	14 × 8;	*i)*	63,125 – 4,125;	*n)*	32 : 2;
e)	63 : 3;	*j)*	3,0625 × 2,125;	*o)*	32 : 16;

p) 401 × 8; *r)* 401 × 4; *t)* 401 × 2;

q) 32 : 8; *s)* 32 : 4; *u)* 933 : 3.

Numerele analizate sunt scrise în sistemul zecimal.

❸ Elaborați un program pentru adunarea şi scăderea numerelor binare.

❹ Elaborați un program pentru înmulțirea şi împărțirea numerelor binare.

3.5. Reprezentarea numerelor naturale în calculator

Calculatoarele actuale utilizează sistemul de numerație binar. Reprezentarea numerelor naturale $N = \{0, 1, 2, ...\}$ se realizează pe un număr fix de poziții binare, de regulă, 8, 16, 32 sau 64 (*fig. 3.1*).

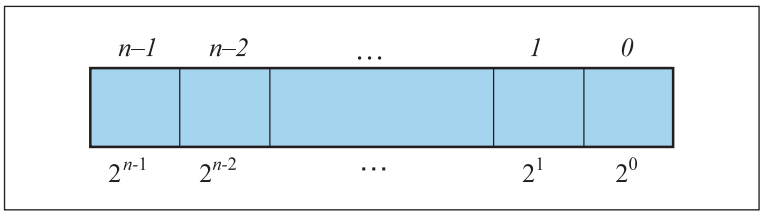


Fig. 3.1. Reprezentarea numerelor naturale pe n poziţii binare

În pozițiile 0, 1, ..., n–1 sunt înscrise cifrele binare ale numărului natural reprezentat în sistemul de numerație binar. Alinierea numerelor binare se realizează la dreapta, eventualele zerouri nesemnificative sunt plasate în fața numărului binar.

Drept exemplu, în *figura 3.2* este redată reprezentarea numărului natural

$$1039 = (10000001111)_2$$

pe un număr de 16 poziții binare.

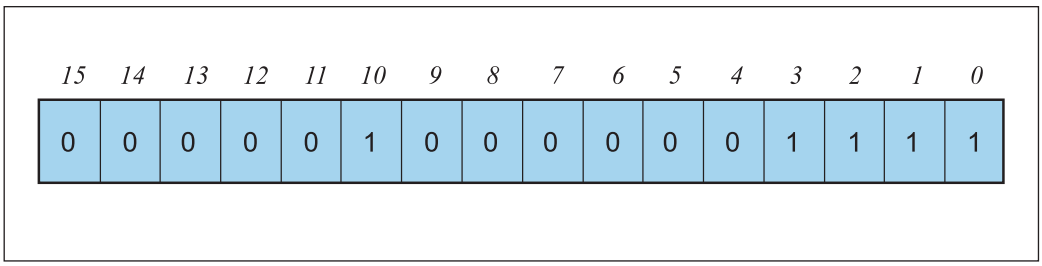


Fig. 3.2. Reprezentarea numărului natural 1039 pe 16 poziţii binare

Numărul maxim ce poate fi reprezentat pe n poziții binare (*fig. 3.3*) este

$$1 \cdot 2^0 + 1 \cdot 2^1 + 1 \cdot 2^2 + ... + 1 \cdot 2^{n-1} = 2^n - 1.$$

Prin urmare, pe n poziții binare pot fi reprezentate numere naturale din intervalul $[0; 2^n-1]$.

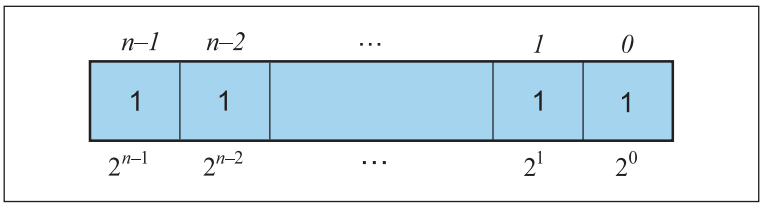


Fig. 3.3. Reprezentarea numărului natural maxim pe n poziţii binare

Întrebări şi exerciţii

❶ Cum se reprezintă numerele naturale în calculator?

❷ Reprezentaţi numerele naturale 3, 112, 191, 204, 255 pe 8 poziţii binare.

❸ Reprezentaţi numerele naturale 3, 255, 1 024, 2 048, 4 096, 65 535 pe 16 poziţii binare.

❹ Calculaţi numerele naturale maxime ce pot fi reprezentate pe 4, 8, 12, 16, 24, 32 şi 64 de poziţii binare.

❺ CERCETEAZĂ! Aflaţi pe câte poziţii binare sunt reprezentate numerele naturale în echipamentele digitale cu care lucraţi.

3.6. Reprezentarea numerelor întregi

În calculator nu există posibilitatea introducerii directe a semnelor + şi –, ataşate numerelor pozitive şi negative. Din acest motiv, reprezentarea semnului numărului x se face cu ajutorul unei cifre binare, denumită **cifră-semn**, aşezată în poziţia n–1 (*fig. 3.4*):

$$S = \begin{cases} 0, & x > 0; \\ 1, & x < 0. \end{cases}$$

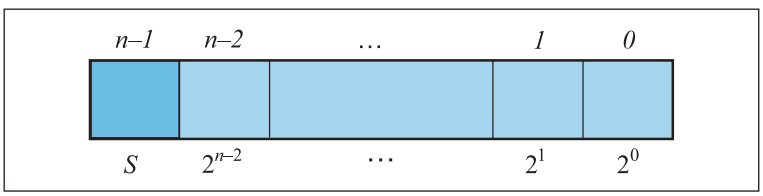


Fig. 3.4. Reprezentarea numerelor întregi pe n poziţii binare

Numerele cu semn se reprezintă cel mai des nu în sistemul binar direct, ci într-un sistem binar codificat care oferă anumite avantaje în executarea operaţiilor aritmetice cu numere algebrice. Din acest punct de vedere, se cunosc trei moduri de reprezentare, denumite **coduri binare pentru numere algebrice**.

Codul direct (codul mărime şi semn). Scrierea unui număr în acest cod este foarte simplă: în cifra-semn se scrie 0, dacă numărul este pozitiv, şi 1, dacă el este negativ; în partea de valoare se înscrie numărul în sistemul binar obişnuit.

Drept exemplu, în *figura 3.5*, p. 126 sunt redate reprezentările numerelor +52 şi –52 pe 8 poziţii binare.

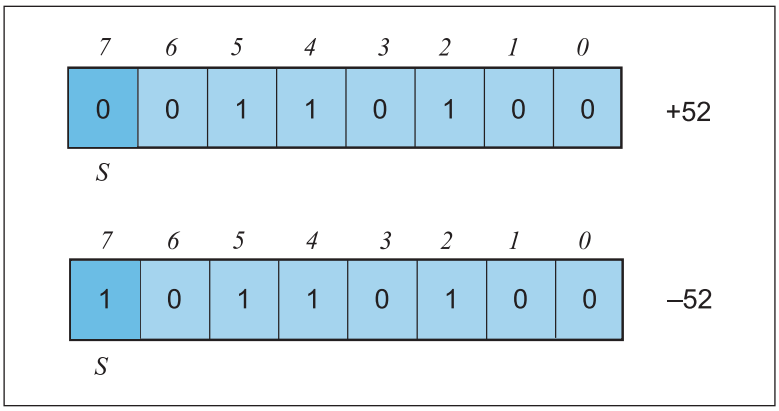


Fig. 3.5. Reprezentarea numerelor +52 şi –52 în cod direct

În codul direct, pe n poziții binare se pot reprezenta numere întregi pozitive şi negative N, astfel încât:

$$-(2^{n-1}-1) \le N \le (2^{n-1}-1).$$

Menționăm că în codul direct există două reprezentări binare pentru numărul 0: 00...0 şi 10...0. Codul direct este rar utilizat în calculatoare, deoarece necesită algoritmi complicați de executare a operațiilor aritmetice şi verificare a rezultatelor.

Codul invers. Pentru numerele pozitive scrierea în cod invers este identică cu cea din codul direct. Dacă numărul este negativ, el se scrie mai întâi ca şi cum ar fi pozitiv, apoi se inversează fiecare cifră binară, adică 1 devine 0 şi 0 devine 1.

În *figura 3.6* sunt redate reprezentările numerelor +52 şi –52 în cod invers pe 8 poziții binare.

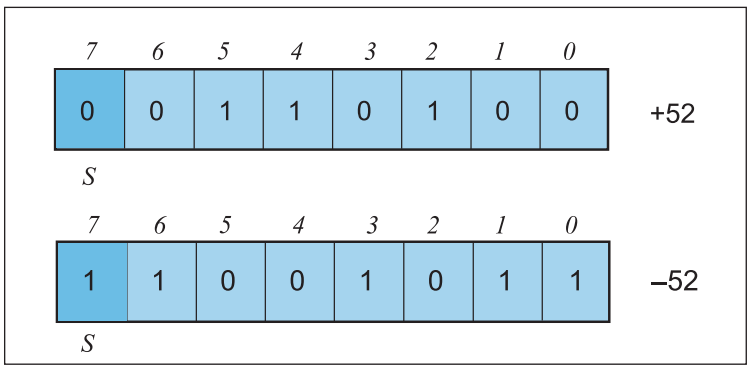


Fig. 3.6. Reprezentarea numerelor +52 şi –52 în cod invers

Se poate uşor constata că intervalul numerelor întregi N care se pot reprezenta în cod invers este acelaşi ca şi pentru reprezentarea în cod direct, iar numărul 0 are două reprezentări binare: 00...0 şi 11...1.

Codul complementar. În acest cod numerele pozitive au aceeaşi reprezentare ca şi în codul direct şi codul invers. Dacă numărul este negativ, el se scrie mai întâi în codul invers, apoi se adună 1 la cifra cea mai puțin semnificativă (poziția binară 0).

Drept exemplu, în *figura 3.7* sunt redate reprezentările numerelor +52 şi –52 în cod complementar.

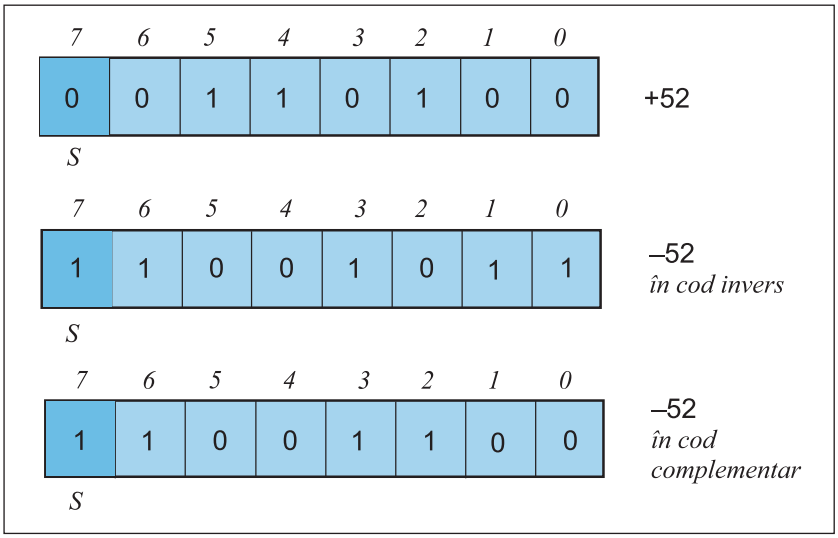


Fig. 3.7. Reprezentarea numerelor +52 şi –52 în cod complementar

Codul complementar este utilizat în marea majoritate a calculatoarelor datorită facilităților pe care le oferă la efectuarea operațiilor aritmetice şi uşurinței cu care se face verificarea rezultatelor. În acest cod pe n poziții binare pot fi reprezentate numere întregi din intervalul $[-2^{n-1}, 2^{n-1} - 1]$.

Întrebări şi exerciţii

❶ Cum pot fi reprezentate în calculator numerele întregi? Explicați cum se scriu numerele negative în cod direct, cod invers şi cod complementar.

❷ Reprezentați în cod direct pe 8 poziții binare:

a) +12; *d)* –12; *g)* +21;

b) –21; *e)* –64; *h)* –68;

c) +68; *f)* +105; *i)* –112.

❸ Reprezentați în cod invers pe 8 poziții binare:

a) +10; *d)* –10; *g)* +65;

b) –65; *e)* +101; *h)* –101;

c) +112; *f)* –112; *i)* –105.

❹ Reprezentați în cod complementar pe 8 poziții binare:

a) +40; *c)* +109; *e)* –40;

b) +27; *d)* –16; *f)* –27;

g)	–109;	*i)*	+16;	*k)*	+111;
h)	–101;	*j)*	+101;	*l)*	–111.

❺ Elaborați un program care afișează pe ecran reprezentările în cod direct, cod invers și cod complementar ale numerelor întregi introduse de la tastatură pentru $n = 8$, 16 și 32.

❻ Cum se reprezintă numărul 0 în cod direct, cod invers și cod complementar? Câte reprezentări are numărul 0 în codurile examinate?

❼ Determinați numărul maxim ce poate fi reprezentat pe n poziții binare în cod direct, cod invers și cod complementar.

❽ CERCETEAZĂ! Aflați pe câte poziții și ce coduri binare pentru numere algebrice sunt utilizate pentru reprezentare numerele întregi în echipamentele digitale cu care lucrați.

3.7. Reprezentarea numerelor reale

Numerele reale se reprezintă în calculator sub formă fracționară prin intermediul reprezentării în virgulă fixă sau în virgulă mobilă (virgulă flotantă).

Reprezentarea în virgulă fixă. În acest caz se consideră că toate numerele au virgula plasată în aceeași poziție, chiar dacă acest lucru nu corespunde formei externe de reprezentare. Procesul de translatare din forma externă în forma internă și invers se realizează cu ajutorul unor coeficienți de scalare aleși în mod corespunzător de programator.

De obicei, se consideră că virgula este plasată imediat după poziția cifrei-semn, caz în care numerele sunt fracții pure (*fig. 3.8*).

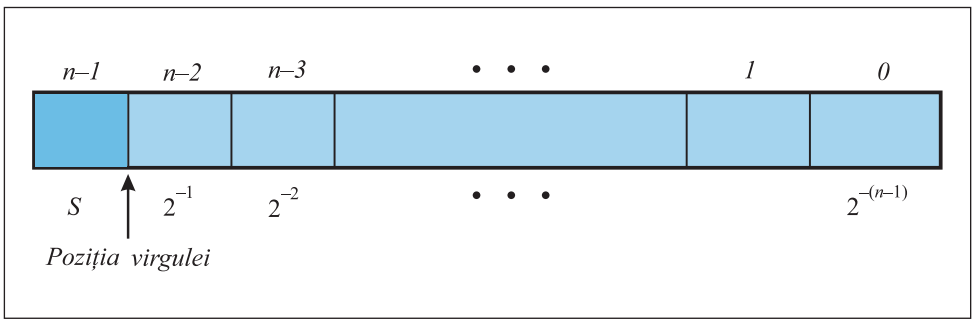


Fig. 3.8. Reprezentarea numerelor reale în virgulă fixă

Virgula însăși nu este materializată fizic în calculator. Din *figura 3.8* rezultă că pe n poziții binare pot fi reprezentate numere reale, valoarea absolută a cărora este

$$0{,}00\ldots0 \le |x| \le 0{,}11\ldots1$$

sau, în sistemul zecimal,

$$0 \le |x| \le 1 - 2^{-(n-1)}.$$

Ca şi în cazul numerelor întregi, numerele subunitare cu semn pot fi reprezentate, cu unele modificări, în cod direct, cod invers sau cod complementar.

Drept exemplu, în *figura 3.9* este redată reprezentarea numerelor

$$+ 0{,}9375 = + 15/16 = (0{,}1111)_2,$$
$$- 0{,}9375 = - 15/16 = (-0{,}1111)_2$$

în virgulă fixă pe 8 poziții binare în cod direct.

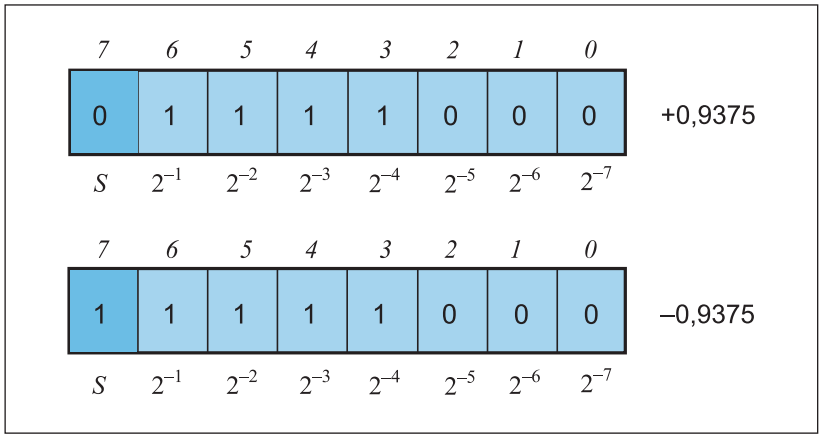


Fig. 3.9. Reprezentarea numerelor +0,9375 şi −0,9375 în virgulă fixă în cod direct

Avantajul principal al reprezentării în virgulă fixă constă în faptul că operațiile aritmetice cu numere reale pot fi efectuate de dispozitivul aritmetic destinat operării cu numere întregi.

Reprezentarea în virgulă mobilă. Operațiile în virgulă fixă sunt comode pentru circuitele omogene de date, când toate numerele reale sunt subunitare. Însă această reprezentare este ineficientă în calculele ştiințifice, unde se lucrează simultan cu numere foarte mari şi numere foarte mici al căror ordin de mărime adesea este imprevizibil. Pentru astfel de probleme se utilizează reprezentarea în virgulă mobilă.

Numerele reprezentate în virgulă mobilă pot fi numere întregi sau fracționare a căror valoare este dată de relația:

$$\boldsymbol{x = M \times b^E},$$

unde b este valoarea bazei, M este un număr subunitar numit **mantisă**, iar E este un **exponent**. În calculatoarele actuale se utilizează $b = 2$ sau 16.

Numărul reprezentat în virgulă mobilă este **normalizat** dacă prima cifră după virgulă a mantisei este diferită de zero.

Exemple:

1) Numărul 23 se va exprima în virgulă mobilă astfel:

$$23 = (10111)_2 = 0{,}10111 \times 2^5,$$

unde $M = 0{,}10111$; $b = 2$; $E = 5$.

2) Numărul 4,9375 se scrie în virgulă mobilă în felul următor:

$$4{,}9375 = (100{,}1111)_2 = 0{,}1001111 \times 2^3,$$

$M = 0{,}1001111$; $b = 2$; $E = 3$.

3) Numărul – 0,375 se scrie în virgulă mobilă după cum urmează:

$$-0{,}375 = (-0{,}011)_2 = -0{,}11 \times 2^{-1},$$

$M = -0{,}11$; $b = 2$; $E = -1$.

Se observă că poziția reală a virgulei în cadrul numărului depinde de valoarea exponentului, adică virgula este mobilă (flotantă).

Există mai multe variante de reprezentare a mantisei şi a exponentului pe n poziții binare. În *figura 3.10* este redată reprezentarea în virgulă mobilă în **formatul exponent-mantisă**.

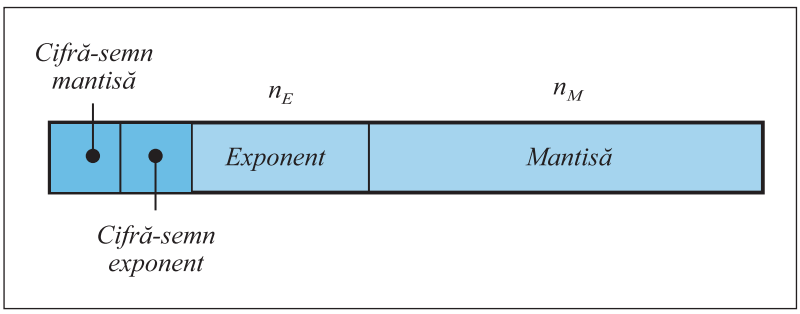


Fig. 3.10. Reprezentarea în virgulă mobilă, formatul exponent-mantisă

Numărul pozițiilor binare n_E alocate exponentului determină domeniul de mărime al numerelor care pot fi reprezentate, în timp ce numărul biților pentru mantisă n_M determină precizia de exprimare a numărului. În calculatoarele actuale $n_E = 6...15$ şi $n_M = 24...64$, fapt ce permite reprezentarea numerelor într-un domeniu foarte larg al ordinului de mărime.

În *figura 3.11* este redată reprezentarea numerelor în formatul **caracteristică-mantisă**.

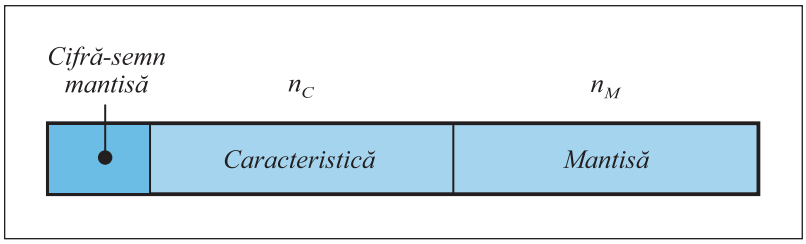


Fig. 3.11. Reprezentarea în virgulă mobilă, formatul caracteristică-mantisă

Caracteristica C constituie o formă de exprimare a exponentului E şi se determină din relația

$$\boldsymbol{C = E + K}.$$

În calculatoarele personale

$$K = 2^{n_c - 1} - 1,$$

unde n_C este numărul de poziții binare alocate caracteristicii. În această reprezentare caracteristica, spre deosebire de exponent, nu poate lua valori negative, fapt ce simplifică structura dispozitivelor aritmetice.

În particular, pentru $n = 8$, caracteristica este calculată astfel:

$$C = E + 127,$$

indicând practic semnul exponentului:

– dacă $C \geq 127$, atunci $E \geq 0$;

– dacă $C < 127$, atunci $E < 0$.

În *tabelul 3.5* sunt prezentate formatele caracteristică-mantisă, utilizate în majoritatea calculatoarelor personale.

Tabelul 3.5

Formatele caracteristică-mantisă în calculatoarele personale

Denumirea formatului	n	n_C	n_M	Precizia mantisei, cifre zecimale	Domeniul de mărime al numerelor
Simplă precizie	32	8	23	6 sau 7	$10^{-37}...10^{38}$
Dublă precizie	64	11	52	15 sau 16	$10^{-307}...10^{308}$
Precizie extinsă	80	15	64	19	$10^{-4932}...10^{4932}$

Întrucât, conform condiției de normalizare, prima cifră a mantisei întotdeauna este cifra 1, acest bit poate să se reprezinte (să ocupe o poziție) sau nu în memoria calculatorului. În cazul în care nu se reprezintă, acest bit se numeşte **bit ascuns**. Accentuăm că tehnica bitului ascuns se referă doar la reprezentarea numerelor în memoria calculatorului, nu şi la operațiile efectuate de dispozitivul aritmetic.

Întrebări şi exerciţii

❶ Care este deosebirea dintre reprezentarea în virgulă fixă şi reprezentarea în virgulă mobilă?

❷ Reprezentați în virgulă fixă pe 8 poziții binare în cod direct:

a) +0,875; *e)* –0,875; *i)* +0,125;

b) –0,125; *f)* +0,3; *j)* –0,3;

c) +0,4; *g)* –0,4; *k)* +0,15625;

d) –0,15625; *h)* +0,21875; *l)* –0,21875.

❸ Care sunt avantajele şi dezavantajele reprezentării în virgulă fixă?

❹ Cum se reprezintă numerele reale în virgulă mobilă? Care numere reprezentate în virgulă mobilă respectă condiția de normalizare?

❺ Cum se efectuează normalizarea numerelor reprezentate în virgulă mobilă?

❻ Exprimați în virgulă mobilă următoarele numere:

a)	+1,5;	*e)*	–1,5;	*i)*	–0,0625;
b)	+0,0625;	*f)*	+3,25;	*j)*	+2,7;
c)	–3,25;	*g)*	–32,87;	*k)*	–2,7;
d)	–6,25;	*h)*	+6,25;	*l)*	–0,147.

❼ De ce depinde precizia şi domeniul de mărime ale numerelor reprezentate în virgulă mobilă?

❽ Care este deosebirea dintre formatele exponent-mantisă şi caracteristică-mantisă? Cum se calculează caracteristicile numerelor reprezentate în virgulă mobilă?

❾ Calculați caracteristicile numerelor din exercițiul 6. Se consideră că $n_C = 8$.

❿ Explicați termenul „bit ascuns”. Care sunt avantajele acestei reprezentări?

⓫ CERCETEAZĂ! Aflați cum se reprezintă numerele naturale, întregi şi reale în calculatorul şi echipamentele digitale la/cu care lucrați dvs.? Determinați numărul de poziții binare alocate fiecărei reprezentări.

Capitolul 4

ALGEBRA BOOLEANĂ

4.1. Variabile şi expresii logice

Algebra booleană sau **algebra logică** este un compartiment al matematicii în care legile gândirii – obiectul de studiu al logicii clasice – sunt studiate cu ajutorul metodelor simbolice. Denumirea aceasta a fost dată în onoarea matematicianului englez George Boole, care în lucrarea *The Laws of Thought* („Legile gândirii"), publicată în 1853, a pus bazele acestei algebre. Mulți ani algebra booleană a fost considerată drept o simplă curiozitate matematică, fără a i se găsi o utilizare într-un anumit domeniu al științelor aplicate. Ea a revenit în actualitate odată cu apariția centralelor telefonice automate şi a calculatoarelor numerice.

Din punct de vedere formal, algebra booleană poate fi definită printr-o mulțime a elementelor {0, 1}, o mulțime a operatorilor elementari $\{\bar{}, \&, \vee\}$ şi printr-un set de postulate. Prin urmare, în algebra booleană orice variabilă nu poate avea decât una dintre cele două valori posibile, notate simbolic prin 0 şi 1, alte valori neavând nicio semnificație.

Variabilele algebrei booleene se notează prin $x, y, z, x_1, x_2, \ldots$ cu sau fără indicii, iar elementele 0 şi 1 se numesc **constante logice**.

Operatorii elementari ai algebrei booleene au următoarele denumiri:

$\bar{}$ – negația (inversia, operația logică *NU*);

& – conjuncția (produsul logic, operația logică *ŞI*);

$\vee$ – disjuncția (suma logică, operația logică *SAU*).

Operatorii elementari se definesc cu ajutorul unor tabele speciale, numite **tabele de adevăr**.

> ***Tabelul de adevăr este un tabel care include toate combinațiile posibile ale valorilor variabilelor față de care este definit operatorul şi rezultatul operației respective.***

În *figura 4.1* sunt prezentate tabelele de adevăr ale negației, conjuncției şi disjuncției.

x	$\bar{x}$
0	1
1	0

x	y	$x \,\&\, y$
0	0	0
0	1	0
1	0	0
1	1	1

x	y	$x \vee y$
0	0	0
0	1	1
1	0	1
1	1	1

Fig. 4.1. Tabelele de adevăr ale operatorilor elementari

Menționăm că în algebra booleană negația se indică printr-o linie orizontală amplasată deasupra variabilei respective.

Întrucât variabila x poate avea numai valoarea 0 sau 1, tabelul de adevăr al negației conține două rânduri. În cazurile conjuncției și disjuncției tabelele de adevăr conțin patru rânduri, câte un rând pentru fiecare din combinațiile posibile 00, 01, 10 și 11 ale variabilelor x și y.

Variabilele și constantele logice, reunite cu ajutorul operatorilor $\overline{\ }$, & și $\vee$ formează **expresii logice**, de exemplu:

1) $x \,\&\, y \vee z$;
2) $x \vee y \,\&\, z$;
3) $\overline{x} \,\&\, y \vee z$;
4) $1 \,\&\, x \vee y$;
5) $0 \vee x \vee y$.
6) $1 \vee 0$.

Valorile expresiilor logice pot fi calculate utilizând tabelele de adevăr ale operatorilor elementari. Pentru evaluarea expresiilor se stabilește următoarea **prioritate a operațiilor logice**:

1) negația;
2) conjuncția;
3) disjuncția.

De exemplu, în expresia

$$x \vee y \,\&\, z$$

mai întâi se execută operația &, iar după aceea operația $\vee$. În expresia

$$x \,\&\, y \vee \overline{z}$$

se execută negația, în continuare conjuncția și, în sfârșit, disjuncția.

Ordinea executării operațiilor logice poate fi schimbată cu ajutorul parantezelor „(” și „)”. Evident, în primul rând, se execută operațiile dintre paranteze.

De exemplu, în cazul expresiei logice

$$x \vee y \,\&\, \overline{x} \vee z$$

se execută negația, conjuncția și disjuncțiile respective, iar în cazul expresiei

$$(x \vee y) \,\&\, (\overline{x} \vee z)$$

după negație se vor executa disjuncțiile și, pe urmă, conjuncția.

Pentru a sistematiza calculele, evaluarea expresiilor logice se efectuează în tabele speciale, numite **tabele de adevăr ale expresiilor logice**.

Tabelul de adevăr al expresiei logice include toate combinațiile posibile ale valorilor variabilelor din expresia examinată și rezultatele operațiilor logice în ordinea calculării lor.

De exemplu, în *figura 4.2* este reprezentat tabelul de adevăr al expresiei,

$$\overline{x} \,\&\, y \vee z,$$

iar în *figura 4.3* – tabelul de adevăr al expresiei

$$\overline{x \,\&\, y \vee z}.$$

x	y	z	$\overline{x}$	$\overline{x}\ \&\ y$	$\overline{x}\ \&\ y \vee z$
0	0	0	1	0	0
0	0	1	1	0	1
0	1	0	1	1	1
0	1	1	1	1	1
1	0	0	0	0	0
1	0	1	0	0	1
1	1	0	0	0	0
1	1	1	0	0	1

Fig. 4.2. Tabelul de adevăr al expresiei logice $\overline{x}\ \&\ y \vee z$

x	y	z	$x\ \&\ y$	$x\ \&\ y \vee z$	$\overline{x\ \&\ y \vee z}$
0	0	0	0	0	1
0	0	1	0	1	0
0	1	0	0	0	1
0	1	1	0	1	0
1	0	0	0	0	1
1	0	1	0	1	0
1	1	0	1	1	0
1	1	1	1	1	0

Fig. 4.3. Tabelul de adevăr al expresiei logice $\overline{x\ \&\ y \vee z}$

Menționăm că, pentru a simplifica notațiile, se admite ca simbolul operatorului & să fie omis din expresiile logice. De exemplu, expresiile logice

$$\overline{x}\ \&\ y \vee z,$$
$$(x \vee y)\ \&\ (\overline{x} \vee z)$$

pot fi scrise şi în forma:

$$\overline{x}y \vee z,$$
$$(x \vee y)(\overline{x} \vee z).$$

Întrebări şi exerciţii

❶ Ce valori poate avea o variabilă logică?
❷ Care sunt operatorii elementari ai algebrei booleene şi cum se definesc ei?
❸ Memorizați tabelele de adevăr ale negației, conjuncției şi disjuncției.
❹ Cum se formează expresiile logice? Care este prioritatea operațiilor logice?
❺ Explicați rolul parantezelor din componența expresiilor logice.

❻ Alcătuiți tabelele de adevăr ale următoarelor expresii logice:

a) $\overline{x}y$;
b) $\overline{x} \vee y$;
c) $\overline{x \vee y}$;
d) $\overline{\overline{x}}$;
e) $\overline{x}x$;
f) $\overline{xy}$;
g) $\overline{x} \vee x$;
h) $\overline{x} \vee \overline{y}$;
i) xyz;
j) $x \vee y \vee z$;
k) $xy \vee z$;
l) $x\,(y \vee z)$;
m) $\overline{x} \vee y \vee z$;
n) $\overline{x}\,(y \vee z)$.

❼ Elaborați un program care introduce valorile logice ale variabilelor *x, y, z* şi afişează valorile uneia dintre expresiile ce urmează:

a) $\overline{x}$;
b) xy;
c) $x \vee y$;
d) $(\overline{x \vee y})\,z$;
e) $\overline{xy} \vee z$;
f) $x \vee \overline{y} \vee z$;
g) $xy \vee z$;
h) $\overline{x \vee y \vee z}$;
i) $x \vee xy$;
j) $\overline{x} \vee y \vee z$.

❽ Elaborați un program care afişează tabelele de adevăr ale conjuncției şi disjuncției.

❾ Elaborați un algoritm repetitiv care formează toate combinațiile posibile ale valorilor variabilelor *x, y* şi *z*. Afişați combinațiile respective pe ecran.

❿ Pentru fiecare dintre expresiile logice ce urmează elaborați câte un program care alcătuieşte tabelul de adevăr respectiv:

a) $x \vee \overline{x}$;
b) $x\overline{x}$;
c) $\overline{x \vee y}$;
d) $\overline{xy}$;
e) $\overline{x} \vee y$;
f) $\overline{x}y$;
g) $x \vee y \vee z$;
h) xyz;
i) $(x \vee y)(\overline{x} \vee y)$;
j) $(x \vee y)(x \vee \overline{y})$.

⓫ CREEAZĂ! Colectați informațiile necesare şi scrieți un eseu despre viața, studiile şi cariera profesională a matematicianului englez *George Boole*.

⓬ DESCOPERĂ! Spre deosebire de logica binară, adică logica ce operează doar cu două valori posibile, există şi extensiuni ale acesteia, care operează cu trei şi chiar cu mai multe valori. Explorând diverse surse de informații, aflați aceste extensiuni şi domeniile în care ele sunt utilizate.

4.2. Funcţii logice

Noţiunea de **funcţie logică** sau **funcţie booleană** se defineşte în acelaşi mod ca şi în cazul algebrei clasice.

Vom nota prin $x_1, x_2, ..., x_n$ un grup arbitrar de variabile booleene, unde $n = 1, 2, 3, ...$. Întrucât fiecare variabilă booleană poate avea numai valorile 0 sau 1, numărul tuturor combinaţiilor posibile ale valorilor variabilelor $x_1, x_2, ..., x_n$ este de 2^n.

Fireşte, pentru $n = 1$ avem 2 combinaţii (0 şi 1); pentru $n = 2$ sunt $2^2 = 4$ combinaţii (00, 01, 10 şi 11); pentru $n = 3$ există $2^3 = 8$ combinaţii (000, 001, 010, 011, 100, 101, 110 şi 111) etc.

Funcţia logică de n variabile $y = f(x_1, x_2, ..., x_n)$ ***este o aplicaţie care pune în corespondenţă fiecărei combinaţii de valori ale variabilelor*** $x_1, x_2, ..., x_n$ ***valoarea 0 sau 1 a variabilei y.***

Variabilele $x_1, x_2, ..., x_n$ se numesc **variabile independente** sau **argumente**, iar variabila y – **variabilă dependentă** sau **funcţie** de argumente $x_1, x_2, ..., x_n$.

Prin urmare, **domeniul de definiţie** al funcţiei $y = f(x_1, x_2, ..., x_n)$ este mulţimea tuturor combinaţiilor posibile

0 0 ... 0
0 0 ... 1
...
1 1 ... 1

ale valorilor argumentelor $x_1, x_2, ..., x_n$, în total 2^n combinaţii, iar **domeniul valorilor** funcţiei logice este mulţimea {0, 1}.

Ca şi în cazul algebrei clasice, funcţiile logice pot fi definite prin **tabele**, **formule** şi **metode grafice**.

Tabelul de adevăr al funcţiei logice $y = f(x_1, x_2, ..., x_n)$ ***este un tabel care include toate combinaţiile posibile ale valorilor argumentelor*** $x_1, x_2, ..., x_n$ ***şi valorile corespunzătoare ale variabilei dependente y.***

De exemplu, în *figura 4.4* este prezentat tabelul de adevăr al unei funcţii logice de 3 variabile.

x_1	x_2	x_3	y
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	0

Fig. 4.4. Tabelul de adevăr al unei funcţii logice de 3 variabile

Tabelul include $2^3 = 8$ rânduri şi are 2 coloane: prima pentru combinaţiile posibile ale valorilor argumentelor x_1, x_2, x_3 şi a doua pentru valorile corespunzătoare ale variabilei dependente y. Conform tabelului analizat, combinaţiei $x_1 = 0$, $x_2 = 0$, $x_3 = 0$ îi corespunde valoarea $y = f(0,0,0) = 1$; combinaţiei $x_1 = 0$, $x_2 = 0$, $x_3 = 1$ – valoarea $y = f(0,0,1) = 0$ etc.

Definirea funcţiei logice prin formule se face atribuind variabilei dependente y valorile expresiilor logice ce conţin argumentele $x_1, x_2, \ldots, x_n$.

De exemplu,

1) $y = x$;

2) $y = x_1 x_2$;

3) $y = \overline{x}_1 x_2 \vee x_3$;

4) $y = x_1 x_2 \vee x_3$.

Natural, cunoscând formula unei funcţii logice, poate fi calculat tabelul ei de adevăr. De exemplu, tabelul de adevăr al funcţiei

$$y = \overline{x_1 x_2 \vee x_3}$$

va fi cel prezentat în *figura 4.4*. Ca să ne convingem de acest fapt, e suficient să calculăm tabelul de adevăr al expresiei logice

$$\overline{x_1 x_2 \vee x_3}$$

prezentat, prin alte notaţii, în *figura 4.3* din paragraful 4.1.

Există mai multe **metode grafice de definire a funcţiilor logice**. Aceste metode se bazează pe diagramele utilizate în teoria mulţimilor şi se studiază în cursurile avansate de informatică.

Întrebări şi exerciţii

❶ Formulaţi definiţia noţiunii *funcţie logică*.

❷ Care sunt domeniul de definiţie şi domeniul de valori ale unei funcţii logice?

❸ Prin ce metode poate fi definită o funcţie logică de n variabile?

❹ Cum se alcătuieşte tabelul de adevăr al unei funcţii de n variabile? Câte rânduri conţine acest tabel?

❺ Cum poate fi alcătuit tabelul de adevăr al unei funcţii logice atunci când se cunoaşte formula ei?

❻ Funcţia logică de 3 variabile $y = f(x_1, x_2, x_3)$ este definită prin tabelul de adevăr din *figura 4.4*. Numiţi combinaţiile valorilor argumentelor x_1, x_2, x_3 pentru care funcţia dată are valoarea $y = 1$. Numiţi combinaţiile respective pentru valoarea funcţiei $y = 0$.

❼ Alcătuiţi tabelele de adevăr ale următoarelor funcţii logice:

a) $y = x$;

b) $y = \overline{x}$;

c) $y = x_1 x_2$;

d) $y = x_1 \vee x_2$;

e) $y = \overline{x_1 x_2}$;

f) $y = \overline{x_1 \vee x_2}$;

g) $y = \overline{x_1 x_2 x_3}$;

h) $y = x_1(x_2 \vee \overline{x}_3)$;

i) $y = \overline{x}_1 \vee x_2 x_3$;

j) $y = x_1 \vee \overline{x_2 x_3}$.

❽ Elaborați un program care introduce valorile logice ale variabilelor x_1, x_2, x_3, x_4 şi afişează valorile uneia dintre funcțiile ce urmează:

a) $y = x_1x_2 \vee x_3x_4$;
b) $y = (x_1 \vee x_2)(x_3 \vee x_4)$;
c) $y = \overline{x_1x_2x_3x_4}$;
d) $y = \overline{x_1 \vee x_2 \vee x_3 \vee x_4}$;
e) $y = \overline{x_1x_2} \vee \overline{x_3x_4}$;
f) $y = \overline{(x_1 \vee x_2)(x_3 \vee x_4)}$;
g) $y = x_1\overline{x}_2 \vee x_3\overline{x}_4$;
h) $y = \overline{x}_1x_2 \vee \overline{x}_3x_4$;
i) $y = \overline{x}_1\overline{x}_2 \vee \overline{x}_3\overline{x}_4$;
j) $y = x_1x_2x_3x_4 \vee \overline{x}_2x_3x_4$;
k) $y = x_1 \vee x_2x_3 \vee x_4$;
l) $y = x_1 \vee x_2 \vee x_3 \vee \overline{x}_1\overline{x}_2\overline{x}_3\overline{x}_4$.

❾ Elaborați un program care alcătuieşte tabelul de adevăr al uneia dintre funcțiile ce urmează:

a) $y = x$;
b) $y = \overline{x}$;
c) $y = x_1x_2$;
d) $y = x_1 \vee x_2$;
e) $y = \overline{x_1x_2}$;
f) $y = \overline{x}_1 \vee \overline{x}_2$;
g) $y = \overline{x_1 \vee x_2}$;
h) $y = \overline{x}_1\overline{x}_2$;
i) $y = \overline{x_1x_2x_3}$;
j) $y = \overline{x}_1 \vee \overline{x}_2 \vee \overline{x}_3$;
k) $y = \overline{x_1 \vee x_2 \vee x_3}$;
l) $y = \overline{x}_1\overline{x}_2\overline{x}_3$;
m) $y = x_1 \vee x_2 \vee \overline{x}_1\overline{x}_2\overline{x}_3\overline{x}_4$;
n) $y = x_1(x_2 \vee \overline{x}_3 \vee \overline{x}_4)$.

4.3. Funcţii logice frecvent utilizate

Să admitem n variabile independente x_1, x_2, ..., x_n. Apare întrebarea, câte funcții logice de n variabile există în algebra booleană? **Numărul funcțiilor logice posibile** poate fi determinat prin următoarele raționamente.

Întrucât orice funcție logică poate fi definită cu ajutorul tabelului de adevăr, numărul funcțiilor posibile de n variabile coincide cu numărul tabelelor distincte de adevăr.

Pentru a defini o funcție logică, în coloana y a tabelului de adevăr se indică valorile funcției – 0 sau 1 pentru fiecare dintre cele 2^n combinații ale valorilor argumentelor. Întrucât tabelul de adevăr are 2^n rânduri, există

$$m = 2^{2^n}$$

funcții logice de n variabile. Funcțiile respective se notează prin y_j, $j = 0, 1, ..., m-1$.

De exemplu, în cazul în care $n = 1$, există $m = 2^{2^1} = 2^2 = 4$ funcții logice, reprezentate în *figura 4.5*.

x	y_0	y_1	y_2	y_3
0	0	1	0	1
1	0	0	1	1

Fig. 4.5. Funcţii logice de o singură variabilă

Evident,

$$y_0 = f(x) = 0;$$
$$y_1 = f(x) = \overline{x};$$
$$y_2 = f(x) = x;$$
$$y_3 = f(x) = 1.$$

Funcțiile y_0 şi y_3 se numesc **funcția constanta 0** şi, respectiv, **constanta 1**. Funcția y_1 este **funcția logică *NU*** sau **negația**, iar funcția y_2 se numeşte **funcția de repetare**.

Pentru $n = 2$ există

$$m = 2^{2^2} = 2^4 = 16$$

funcții logice, reprezentate în *figura 4.6*.

$x_1\ x_2$	y_0	y_1	y_2	y_3	y_4	y_5	y_6	y_7	y_8	y_9	y_{10}	y_{11}	y_{12}	y_{13}	y_{14}	y_{15}
0 0	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
0 1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
0 1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1 1	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1

Fig. 4.6. Funcţii logice de două variabile

Funcțiile y_0 şi y_{15} sunt funcțiile deja cunoscute, constanta 0 şi, respectiv, constanta 1:

$$y_0 = f(x_1, x_2) = 0;$$
$$y_{15} = f(x_1, x_2) = 1.$$

Funcția y_8 poate fi scrisă în forma:

$$y_8 = f(x_1, x_2) = x_1 x_2.$$

În mod firesc, funcția y_8 va avea denumirea **funcția logică *ŞI*** sau **conjuncția**.

Funcția y_{14} poate fi scrisă în forma:

$$y_{14} = f(x_1, x_2) = \overline{x_1 \vee x_2}.$$

Prin urmare, funcția y_{14} va avea denumirea **funcția logică *SAU*** sau **disjuncția**.

Funcțiile logice NU, ŞI, SAU, induse de operatorii elementari, respectiv, $\overline{\ }$, &, $\vee$ se numesc funcții logice elementare.

Din *figura 4.6* se observă că

$$y_1 = x_1 \vee x_2.$$

Funcția dată se numeşte **funcția logică *SAU-NU***.

În mod similar, funcția y_7 poate fi exprimată în forma

$$y_7 = f(x_1, x_2) = \overline{x_1 x_2}.$$

Funcția în cauză se numeşte **funcția logică *ŞI-NU***.

Funcția

$$y_9 = f(x_1, x_2) = \overline{x}_1 \overline{x}_2 \vee x_1 x_2$$

are valoarea 1 numai când $x_1 = x_2 = 0$ sau $x_1 = x_2 = 1$. Această funcție se numeşte **funcția logică *COINCIDENȚĂ*** sau **echivalență** şi se notează prin simbolul „$\equiv$”.

Analizând tabelul din *figura 4.6*, mai observăm că

$y_3 = f(x_1, x_2) = \overline{x}_1$ (negația variabilei x_1);

$y_{12} = f(x_1, x_2) = x_1$ (repetarea variabilei x_1);

$y_5 = f(x_1, x_2) = \overline{x}_2$ (negația variabilei x_2);

$y_{10} = f(x_1, x_2) = x_2$ (repetarea variabilei x_2).

Pentru a fi mai uşor memorizate, în *figura 4.7* sunt prezentate tabelele de adevăr ale funcțiilor logice *NU, ŞI, SAU, ŞI-NU, SAU-NU* şi *COINCIDENȚĂ*.

NU

x	$\overline{x}$
0	1
1	0

ŞI

x_1	x_2	x_1x_2
0	0	0
0	1	0
1	0	0
1	1	1

SAU

x_1	x_2	$x_1 \vee x_2$
0	0	0
0	1	1
1	0	1
1	1	1

ŞI-NU

x_1	x_2	$\overline{x_1x_2}$
0	0	1
0	1	1
1	0	1
1	1	0

SAU-NU

x_1	x_2	$\overline{x_1 \vee x_2}$
0	0	1
0	1	0
1	0	0
1	1	0

COINCIDENȚĂ

x_1	x_2	$x_1 \vee x_2$
0	0	1
0	1	0
1	0	0
1	1	1

Fig. 4.7. Funcţiile logice frecvent utilizate

În mod similar pot fi studiate funcțiile logice de 3 variabile, numărul cărora este de $m = 2^{2^3} = 2^8 = 256$; funcțiile de 4 variabile, numărul cărora este de $m = 2^{2^4} = 2^{16} = 65\,536$ etc. Se observă că, deşi este finit, numărul funcțiilor booleene posibile este enorm. S-a demonstrat însă că **orice funcție logică de n variabile, $n \geq 2$, poate fi exprimată printr-o formulă care include numai operatorii elementari Ż, &, $\vee$.** Această proprietate facilitează realizarea tehnică a dispozitivelor destinate calculării funcțiilor logice cu un număr arbitrar de argumente.

Întrebări şi exerciţii

❶ Determinați numărul funcțiilor logice de 5 şi de 6 variabile.

❷ Numiți funcțiile logice elementare şi alcătuiți tabelele respective de adevăr.

❸ Memorizați tabelele de adevăr ale funcțiilor logice frecvent utilizate *NU*, *ŞI*, *SAU*, *ŞI-NU*, *SAU-NU* şi *COINCIDENȚĂ*.

❹ Elaborați un program care va afişa pe ecran tabelul de adevăr al uneia dintre funcțiile logice y_0, y_1, y_2, y_3 de 2 variabile.

❺ Elaborați un program care va afişa pe ecran tabelul de adevăr al uneia dintre funcțiile logice y_j de n variabile.

Capitolul 5

CIRCUITE LOGICE

5.1. Circuite logice elementare

Circuitul logic este un dispozitiv destinat calculării funcțiilor logice. Pentru a realiza circuitele logice, e necesar ca valorile binare **0** şi **1** ale argumentelor şi funcțiilor respective să fie reprezentate prin valorile unor mărimi fizice, de exemplu: presiune, temperatură, tensiune sau curent electric, flux luminos etc. În funcție de mărimile fizice utilizate, deosebim dispozitive logice mecanice, pneumatice, hidraulice, electromecanice, electronice, optice etc. În dispozitivele hidraulice şi pneumatice valorile logice **0** sau **1** pot fi reprezentate prin valorile mici şi, respectiv, mari ale presiunii fluidului, în dispozitivele electromecanice şi electronice – prin prezența sau absența curentului electric, prin niveluri de tensiune etc.

Pentru o înțelegere clară a principiilor de funcționare a dispozitivelor logice, vom studia mai întâi circuitele cu contacte. Componentele de bază ale acestor circuite sunt **elementele de comutare** – contactele electrice normal deschise şi contactele electrice normal închise.

În cazul **contactelor normal deschise**, circuitul electric este deschis dacă contactele nu sunt acționate şi – închis la acționarea lor. În cazul **contactelor normal închise**, circuitul electric este închis dacă contactele nu sunt acționate şi – deschis la acționarea lor (*fig. 5.1*).

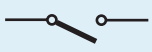
	Neacționate	*Acționate*
Contacte normal deschise		
Contacte normal închise		

Fig. 5.1. Contacte normal deschise şi normal închise

De exemplu, contactele unui întrerupător electric uzual sunt contacte normal deschise, iar contactele butonului de pauză al magnetofonului sunt contacte normal închise.

În circuitele cu contacte, valorile logice ale argumentelor sunt reprezentate prin stările contactelor electrice respective. Valorii logice **1** îi corespunde starea „*contactul este acționat*”, iar valorii logice **0** îi corespunde starea „*contactul este neacționat*”.

Circuitul electric care realizează **funcția logică *NU*** şi simbolul utilizat sunt prezentate în *figura 5.2*.

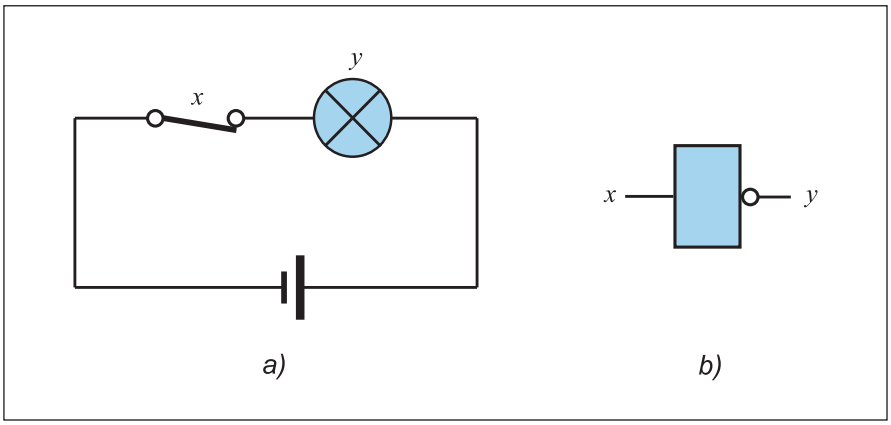


Fig. 5.2. Circuit cu contacte pentru realizarea funcţiei logice *NU* (*a*) şi simbolul utilizat (*b*)

Argumentul x este materializat prin contactul normal închis, iar valorile variabilei dependente y sunt reprezentate prin stările becului electric: *stins* (valoarea logică **0**) sau *aprins* (valoarea logică **1**). Se observă că becul va fi aprins (y=1), dacă contactul normal închis nu este acționat (x=0).

Funcția logică *ŞI* se realizează prin conectarea în serie a contactelor electrice. Circuitul electric care realizează funcția *ŞI* de două variabile şi simbolul utilizat sunt prezentate în *figura 5.3*.

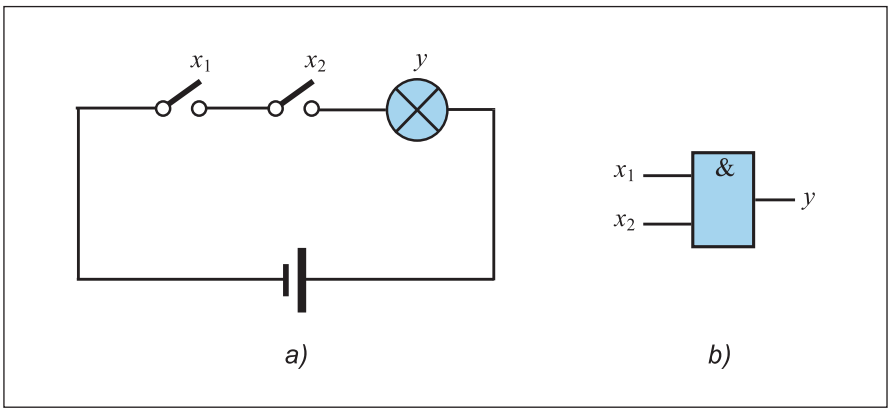


Fig. 5.3. Circuit cu contacte pentru realizarea funcţiei logice *ŞI* (*a*) şi simbolul utilizat (*b*)

Variabilele x_1 şi x_2 sunt materializate prin cele două contacte normal deschise, iar valorile variabilei y prin bec. Se observă că becul va fi aprins (y=1), numai dacă ambele contacte normal deschise sunt acționate (x_1=1 şi x_2=1).

Funcția logică *SAU* se realizează prin conectarea în paralel a contactelor electrice. Circuitul respectiv este prezentat în *figura 5.4*, p. 144.

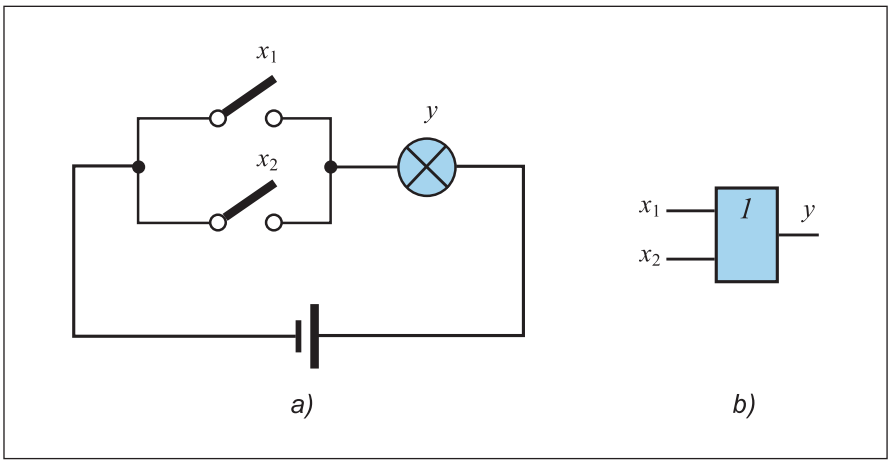


Fig. 5.4. Circuite cu contacte pentru realizarea funcţiei logice *SAU* (*a*) şi simbolul utilizat (*b*)

Se constată că becul va fi aprins (y=1), dacă cel puţin unul dintre cele două contacte normal deschise este acţionat (x_1=1 sau x_2=1).

Întrucât viteza de închidere-deschidere a contactelor electrice este foarte mică, în calculatoarele moderne valorile **0, 1** sunt reprezentate prin niveluri de tensiune, iar ca element de comutare se utilizează tranzistorul.

Tranzistorul este un dispozitiv electronic format în sau pe suprafaţa unui monocristal semiconductor. Tehnologiile avansate permit fabricarea a 10^6–10^7 de tranzistoare pe o suprafaţă de 1 cm^2 al monocristalului.

În regim de comutaţie, tranzistorul poate fi considerat ca un întrerupător obişnuit, care într-o stare conduce curentul (este închis), iar în alta – nu (este deschis). Spre deosebire însă de întrerupătorul obişnuit, închiderea sau deschiderea tranzistorului se realizează cu ajutorul curentului electric.

Există mai multe tipuri de tranzistoare. În *figura 5.5* este prezentat tranzistorul *n-p-n* (abrevierea se referă la structura internă a tranzistorului) şi schemele echivalente în regim de comutaţie.

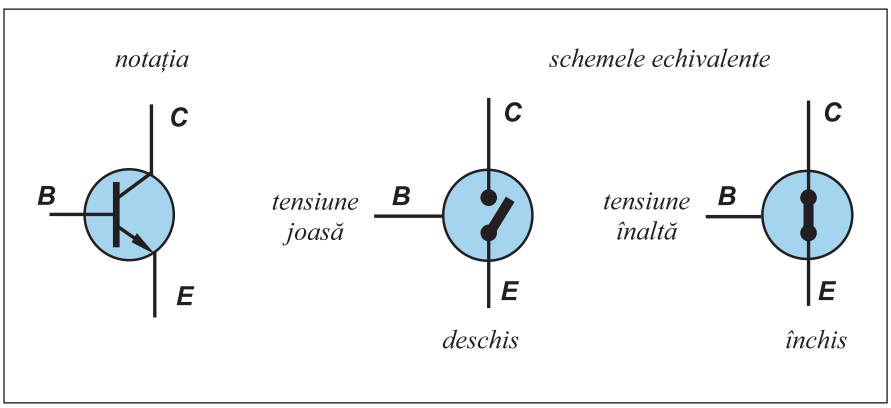


Fig. 5.5. Tranzistorul *n-p-n*

Tranzistorul *n-p-n* are trei terminale: emitorul *E*, baza *B* şi colectorul *C*. În regim de comutaţie, emitorul şi colectorul pot fi consideraţi drept contacte care se închid sau se deschid cu ajutorul unei tensiuni aplicate la bază. Menţionăm că tranzistoarele moderne permit efectuarea a 10^6–10^9 închideri-deschideri pe secundă. Ca şi în cazul contactelor electrice studiate mai sus, utilizarea diferitor tipuri de tranzistoare şi conectarea lor în serie sau în paralel permite realizarea funcţiilor logice *NU*, *ŞI*, *SAU*.

Circuitele destinate calculării funcţiilor logice frecvent utilizate se numesc circuite logice elementare sau porţi logice.

Simbolurile utilizate pentru notarea porţilor logice sunt reprezentate în *figura 5.6*.

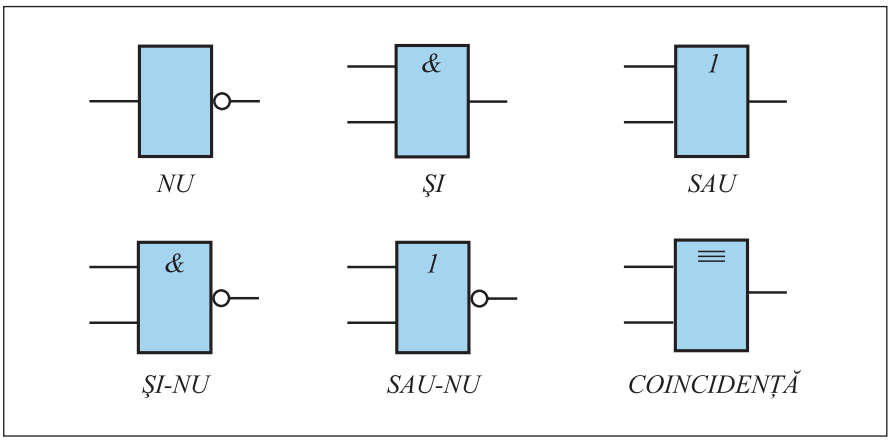


Fig. 5.6. Simbolurile porţilor logice

E cunoscut faptul că orice funcţie logică poate fi exprimată printr-o formulă care include numai simbolurile operatorilor elementari $\overline{\ }$, &, $\vee$. Prin urmare, **orice funcţie logică cu un număr arbitrar de argumente poate fi materializată prin conectarea porţilor logice *NU*, *ŞI*, *SAU*.** De exemplu, funcţia

$$y = x_1x_2 \vee \overline{x}_2x_3$$

poate fi realizată cu ajutorul următoarelor porţi logice:

– o poartă *NU* pentru calcularea $\overline{x}_2$;

– două porţi logice *ŞI* pentru calcularea conjuncţiilor x_1x_2 şi $\overline{x}_2x_3$;

– o poartă *SAU* pentru calcularea disjuncţiei $x_1x_2 \vee \overline{x}_2x$;

Schema circuitului logic pentru calcularea funcţiei respective este prezentată în *figura 5.7*, p. 146.

Întrebări şi exerciţii

❶ Cum pot fi reprezentate valorile binare **0** şi **1**?

❷ Cum funcţionează contactele normal deschise şi contactele normal închise?

❸ Care este reprezentarea valorilor binare **0** şi **1** în circuitele cu contacte?

❹ EXPERIMENTEAZĂ! Utilizând trusa din laboratorul de fizică, montaţi cir-

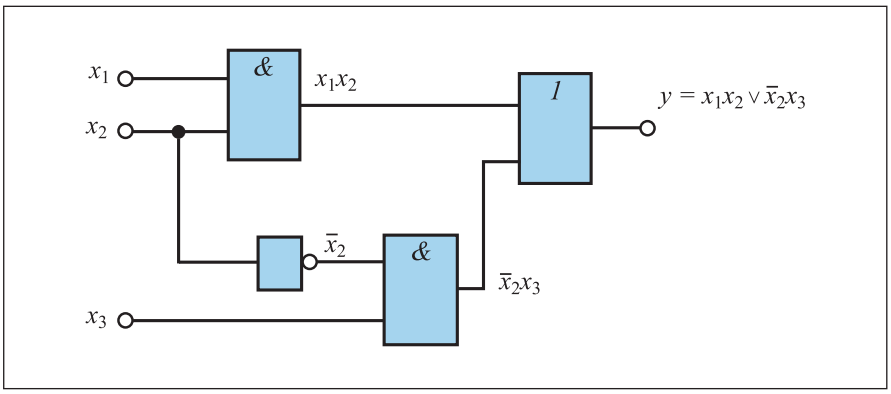


Fig. 5.7. Circuitul logic pentru realizarea funcţiei $y = x_1x_2 \vee \breve{}_2x_3$

cuitele din *figurile 5.2*, *5.3* şi *5.4*. Verificați tabelele de adevăr ale funcțiilor realizate de circuitele în cauză.

❺ Cum se realizează funcțiile *NU*, *ŞI*, *SAU* în cazul circuitelor cu contacte electrice?

❻ Care este rolul elementelor de comutare la realizarea circuitelor logice?

❼ Care este rolul tranzistorului în calculatoarele moderne?

❽ Memorizați simbolurile porților logice. Explicați cum se utilizează porțile logice la realizarea funcțiilor logice arbitrare.

❾ PROIECTEAZĂ! Utilizând porțile *NU*, *ŞI*, *SAU*, elaborați circuitele logice pentru calcularea următoarelor funcții:

a) $y = x_1x_2x_3$;
b) $y = x_1 \vee x_2 \vee x_3$;
c) $y = \bar{x}_1x_2x_3$;
d) $y = \bar{x}_1 \vee x_2 \vee x_3$;
e) $y = x_1x_2 \vee x_3x_4$;
f) $y = (x_1 \vee x_2)(x_3 \vee x_4)$;
g) $y = x_1x_2$;
h) $y = x_1 \vee x_2$;
i) $y = x_1x_2 \vee \overline{x_2x_3}$;
j) $y = (x_1 \vee x_2)(\overline{x_2 \vee x_3})$;
k) $y = x_1x_2 \vee \bar{x}_1x_3 \vee x_3x_4$;
l) $y = \bar{x}_1x_2 \vee \bar{x}_1x_3 \vee x_2x_3$;
m) $y = \bar{x}_1x_2 \vee x_1\bar{x}_2$;
n) $y = x_1x_2 \vee \bar{x}_1\bar{x}_2$;
o) $y = x_1(x_2 \vee x_3 \vee x_4)$;
p) $y = x_1 \vee x_2x_3x_4$.

❿ CERCETEAZĂ! Releul electromagnetic este un dispozitiv cu care se comandă închiderea sau deschiderea contactelor electrice. Contactele respective sunt acționate de un electromagnet. Cum poate fi utilizat releul pentru realizarea funcțiilor logice *NU*, *ŞI*, *SAU*?
EXPERIMENTEAZĂ! Cu piesele din trusa din laboratorul de fizică asamblați un releu electromagnetic şi verificați soluțiile propuse de dvs.

⓫ EXPLOREAZĂ! Reprezentând valorile binare ale variabilelor de ieşire prin prezența (valoarea **1**) sau absența (valoarea **0**) a fluidului, elaborați circuitele hidraulice pentru realizarea funcțiilor logice *NU*, *ŞI*, *SAU*. Montați instalațiile respective utilizând robinetele şi furtunurile din trusa din laboratorul de chimie. Verificați tabelele de adevăr ale funcțiilor logice realizate.

5.2. Clasificarea circuitelor logice

Circuitele logice se clasifică în două categorii: circuite combinaționale şi circuite secvențiale.

Într-un **circuit combinațional** valorile variabilelor de ieşire sunt determinate în orice moment de combinația valorilor variabilelor de intrare conform funcțiilor logice ale circuitului. Într-un **circuit secvențial** valorile variabilelor de ieşire depind nu numai de combinațiile valorilor variabilelor de intrare, dar şi de consecutivitatea aplicării lor. Altfel spus, circuitele combinaționale reprezintă circuite logice lipsite de elemente de memorie, iar circuitele secvențiale includ şi elementele de memorie binară. Prin urmare, un circuit combinațional realizează o prelucrare numerică a informației, care poate fi în întregime exprimată printr-un grup de funcții logice în care nu intervine parametrul timp.

Schema-bloc a unui circuit combinațional este prezentată în *figura 5.8*.

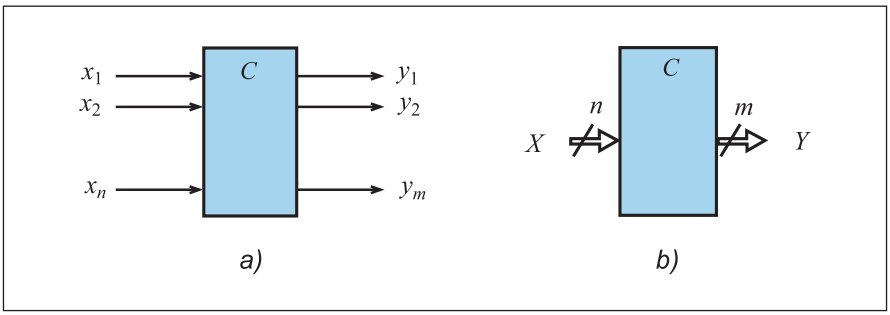


Fig. 5.8. Schema-bloc a unui circuit logic combinaţional:
a – detaliat; *b* – generalizat

Circuitul are n variabile de intrare: $X = < x_1, x_2, ..., x_n >$, şi m variabile de ieşire: $Y = < y_1, y_2, ..., y_m >$. Conexiunile destinate transmiterii valorilor unui grup de variabile se reprezintă prin linii duble. Dacă e necesar, numărul de variabile se indică lângă o bară care întretaie linia dublă a grupului de variabile. De exemplu, în *figura 5.8 b* se indică că grupul X include n variabile, iar grupul Y – m variabile.

Din punctul de vedere al teoriei informației, circuitul combinațional reprezintă un convertor de cod: la intrările X se aplică combinațiile admise de primul cod, din care se face conversiunea, iar la ieşirile Y apar combinațiile corespunzătoare în cel de al doilea cod, în care se face conversiunea.

De exemplu, primul cod poate fi codul *EBCDIC*, iar al doilea codul *ASCII*.

5.3. Sumatorul

Una dintre principalele sarcini ale unui calculator în momentul prelucrării informației constă în efectuarea operațiilor aritmetice elementare şi, în special, adunarea şi scăderea. Dispozitivele în care au loc aceste operații au la bază circuite cu ajutorul cărora se efectuează adunarea, respectiv scăderea a două cifre binare.

Semisumatorul este un circuit combinațional destinat adunării a două cifre binare. Tabelul de adevăr care pune în evidență funcționarea unui semisumator rezultă din regula de adunare a două cifre binare şi este prezentat în *figura 5.9*. Aici a şi b reprezintă cele două cifre binare care se adună, s – cifra-sumă a rangului respectiv, iar t – cifra de transport către rangul următor.

a	b	s	t
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Fig. 5.9. Tabelul de adevăr pentru adunarea a două cifre binare

Pentru a elabora o schemă posibilă a semisumatorului, exprimăm funcțiile de ieşire s şi t:

$$s = \overline{a}b \vee a\overline{b};$$

$$t = ab.$$

Schema care realizează funcțiile s, t şi simbolul utilizat sunt prezentate în *figura 5.10*.

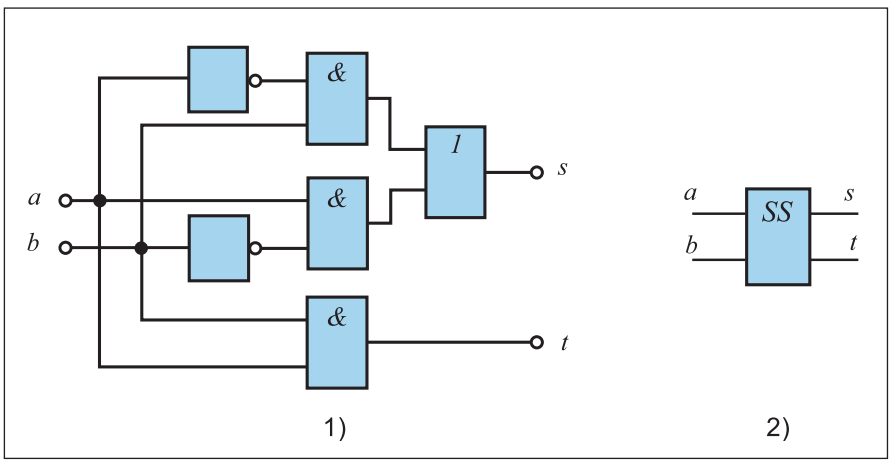


Fig. 5.10. Schema semisumatorului (1) şi simbolul utilizat (2)

Să luăm două numere binare

$$A = a_{n-1}\, a_{n-2} \ldots a_j \ldots a_0$$

şi

$$B = b_{n-1}\, b_{n-2} \ldots b_j \ldots b_0,$$

unde a_j şi b_j reprezintă cifrele binare din rangul (poziția) j. La însumarea cifrelor a_j şi b_j din rangul j trebuie să se ia în considerare şi cifra de transport t_{j-1} de la rangul inferior j–1:

$$\begin{array}{r} \quad\quad\quad\quad\quad t_{j-1} \\ \quad a_{n-1}\ a_{n-2}\ \ldots\ a_j\ \ldots\ a_0 \\ + \\ \quad b_{n-1}\ b_{n-2}\ \ldots\ b_j\ \ldots\ b_0. \end{array}$$

Se obține astfel un circuit combinațional care calculează suma $t_{j-1} + a_j + b_j$, denumit **sumator elementar**.

Sumatorul elementar poate fi realizat prin conectarea în cascadă a două semisumatoare SS_1 şi SS_2 (*fig. 5.11*).

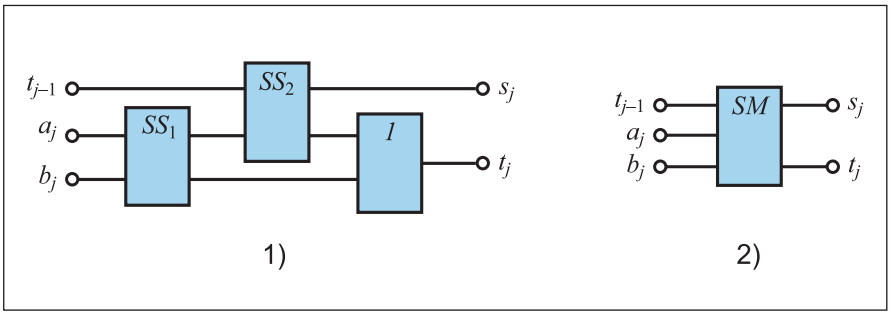


Fig. 5.11. Schema sumatorului elementar (1) şi simbolul utilizat (2)

Semisumatorul SS_1 calculează suma $(a_j + b_j)$, iar semisumatorul SS_2 însumează transportul t_{j-1} cu suma $(a_j + b_j)$ calculată de primul semisumator. Transportul t_j către rangul superior $j+1$ se calculează de poarta logică *SAU,* care reuneşte transporturile intermediare de la ieşirile respective ale semisumatoarelor SS_1 şi SS_2.

Suma numerelor binare A şi B se calculează cu ajutorul unui circuit combinațional denumit **sumator**. Un sumator poate fi realizat prin conectarea a n sumatoare elementare (*fig. 5.12*, p. 150).

Sumatorul elementar SM_0 corespunzător cifrei celei mai puțin semnificative poate fi înlocuit cu un semisumator, deoarece pentru această poziție nu există un transport de la rangul precedent. Transportul de la ieşirea sumatorului SM_{n-1} al rangului cel mai semnificativ este folosit pentru a indica **depăşirea capacității** sumatorului de n biți.

Din analiza *figurilor 5.10*, *5.11* şi *5.12* rezultă că un dispozitiv complex – sumatorul de n biți, este realizat prin reunirea a unor dispozitive mult mai simple, adică a n sumatoare elementare. Fiecare sumator elementar, la rândul lui, este realizat prin reunirea a câte două semisumatoare şi o poartă logică *SAU*.

Metoda de elaborare a dispozitivelor complexe (de exemplu, sumatorul) prin reunirea mai multor dispozitive identice mai simple (sumatorul elementar) poartă denumirea de **metodă de proiectare ierarhică**. Conform acestei metode, componentele calculatorului se caracterizează prin niveluri de ierarhie, de exemplu:

nivelul 1 – tranzistoare;
nivelul 2 – porți logice;
nivelul 3 – semisumatoare, sumatoare elementare etc.;
nivelul 4 – sumatoare, scăzătoare etc.;
nivelul 5 – unități aritmetice, unități de comandă etc.

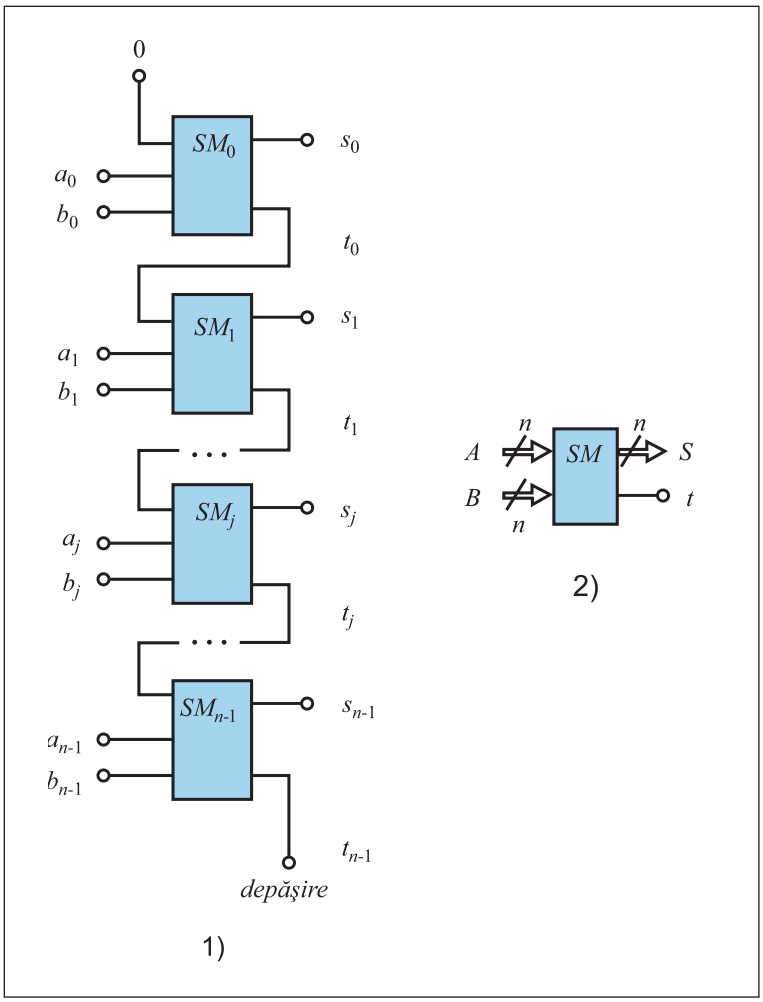


Fig. 5.12. Schema sumatorului (1) şi simbolul utilizat (2)

Componentele de un nivel ierarhic inferior sunt utilizate în calitate de „cubuşoare" elementare pentru realizarea componentelor de un nivel ierarhic superior.

Teoretic, dispozitivele unui calculator numeric pot fi elaborate şi fără aplicarea metodei de proiectare ierarhică. De exemplu, în cazul sumatorului elementar, utilizarea semisumatorului nu este obligatorie. E suficient să alcătuim tabelul de adevăr al sumatorului elementar, să exprimăm funcţiile de ieşire prin formule şi să reunim porţile logice respective.

În cazul nivelurilor ierarhice superioare, neaplicarea metodei de proiectare ierarhică face imposibilă elaborarea dispozitivelor complexe. De exemplu, în cazul unui sumator de n biţi, tabelul respectiv de adevăr ar conţine 2^{2n} rânduri. Pentru $n = 16$ obţinem $2^{2 \cdot 16} = 2^{32} \approx 10^9$ rânduri. Evident, formulele funcţiilor de ieşire ale sumatorului de 16 biţi practic nu mai pot fi scrise. Prin urmare, suntem nevoiţi să aplicăm metoda de proiectare ierarhică şi să realizăm sumatorul printr-o reunire de n sumatoare elementare.

Aplicând metoda de proiectare ierarhică, într-un mod similar pot fi elaborate circuitele combinaționale destinate scăderii numerelor binare: **semiscăzătorul**, **scăzătorul elementar** şi **scăzătorul**.

Întrebări şi exerciţii

❶ Care este destinația semisumatorului? Dar a sumatorului elementar? Şi în fine a sumatorului de n biți?

❷ CREEAZĂ! Alcătuiți tabelul de adevăr al sumatorului elementar. Tabelul va conține cinci coloniță: trei pentru intrările a_j, b_j, t_{j-1} şi două pentru ieşirile s_j, t_j.

❸ Elaborați un program care simulează funcționarea sumatorului elementar. Cifrele binare a_j, b_j şi cifra de transport t_{j-1} de la rangul inferior se citesc de la tastatură, iar cifra-sumă s_j şi cifra de transport t_j către rangul superior se afişează pe ecran.

❹ Explicați esența metodei de proiectare ierarhică a dispozitivelor unui calculator numeric. Este oare obligatorie aplicarea acestei metode? Argumentați răspunsul dvs.

❺ Câte porți logice *NU*, *ŞI*, *SAU* va conține un sumator de 16 biți? Dar de 32 de biți?

❻ PROIECTEAZĂ! **Semiscăzătorul** este un circuit combinațional destinat scăderii a două cifre binare. Circuitul respectiv are intrările a, b şi ieşirile d, i. Prin termenul d se înțelege diferența $a–b$, iar prin termenul i – împrumutul de la cifra de rang imediat superior. Alcătuiți tabelul de adevăr şi elaborați schema semiscăzătorului.

❼ PROIECTEAZĂ! **Scăzătorul elementar** este un circuit combinațional capabil de a calcula diferența d_j şi împrumutul i_j către rangul imediat superior, dacă se introduc la intrare descăzutul a_j, scăzătorul b_j şi împrumutul i_{j-1} de la rangul anterior. Aplicând metoda de proiectare ierarhică, elaborați schema scăzătorului elementar.

❽ PROIECTEAZĂ! Aplicând metoda de proiectare ierarhică, elaborați schema unui **scăzător** de n biți.

❾ Câte porți logice *NU*, *ŞI*, *SAU* va conține un scăzător de 16 biți? Dar de 32 de biți?

❿ ANALIZEAZĂ! Este oare obligatorie aplicarea metodei de proiectare ierarhică în cazul elaborării unui scăzător de n biți? Argumentați răspunsul dvs.

5.4. Circuite combinaţionale frecvent utilizate

Circuitele combinaționale frecvent utilizate sunt prezentate în *figura 5.13*, p. 152.

Sumatorul este un circuit combinațional destinat adunării a două numere binare. Tabelul de adevăr şi schema sumatorului au fost studiate în paragraful precedent.

Comparatorul este un circuit combinațional care compară numerele binare A şi B, indicând prin cele trei ieşiri una dintre situațiile posibile: $A < B$, $A > B$ sau $A = B$.

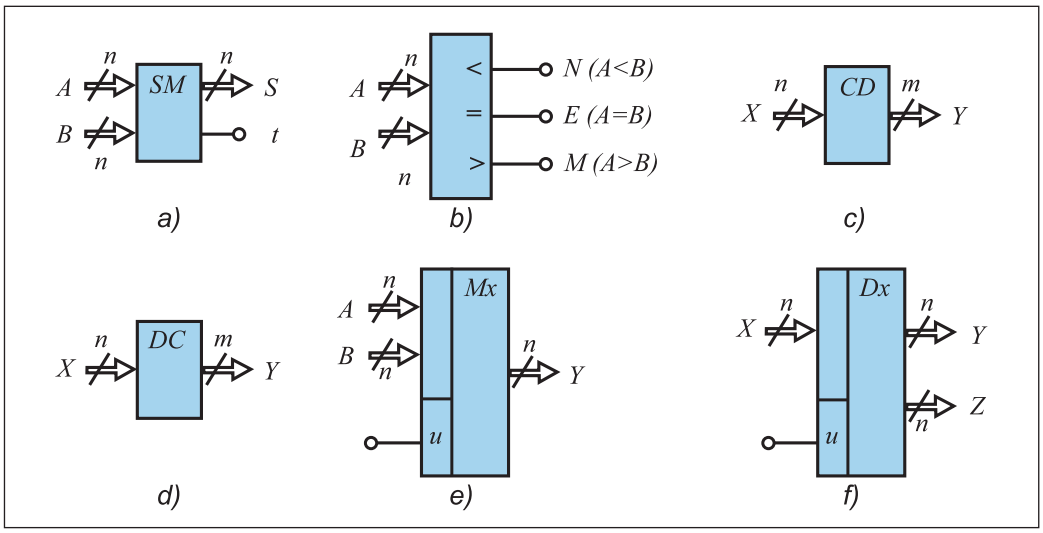


Fig. 5.13. Circuite combinaţionale frecvent utilizate:
a – sumatorul; *b* – comparatorul; *c* – codificatorul; *d* – decodificatorul;
e – multiplexorul; *f* – demultiplexorul

Codificatorul este un circuit combinațional care efectuează conversiunea mesajelor s_1, s_2, ..., s_n în cuvintele binare din codul respectiv. Se consideră că fiecare dintre mesajele s_i este reprezentat prin valorile $x_1 = 0$, ..., $x_i = 1$, ..., $x_n = 0$ aplicate la intrarea codificatorului, iar cuvântul de cod – prin variabilele de ieşire $y_1, y_2, ..., y_m$.

De exemplu, variabilele x_1, x_2, x_3, ... pot reprezenta starea tastelor <*A*>, <*B*>, <*C*>, ... ale tastaturii. Codificatorul respectiv va furniza la ieşire cuvântul de cod *ASCII* corespunzător tastei acționate.

Decodificatorul este un circuit combinațional care generează semnalul logic 1 pe o ieşire distinctă pentru fiecare combinație a valorilor variabilelor de intrare. Cu alte cuvinte, decodificatorul efectuează operația inversă a unui codificator.

Decodificatoarele sunt utilizate pentru a determina operațiile pe care trebuie să le execute procesorul, pentru selectarea unităților de intrare-ieşire, sintetizarea simbolurilor etc.

Multiplexorul este un circuit combinațional destinat selectării fluxurilor de date. În *figura 5.13e* este prezentat un multiplexor care transmite la ieşire biții numărului binar A ($u = 0$) sau B ($u = 1$). În calculatoarele moderne multiplexoarele se utilizează pentru transferul informației de la mai multe surse la un singur destinatar.

Demultiplexorul distribuie fluxul de date de la intrarea X la una dintre ieşirile Y ($u = 0$) sau Z ($u = 1$). Drept exemplu amintim transferul informației de la o singură sursă la mai mulți destinatari.

Întrebări şi exerciţii

❶ Explicați destinația circuitelor combinaționale frecvent utilizate: sumatorul, comparatorul, codificatorul, decodificatorul, multiplexorul şi demultiplexorul.

❷ Alcătuiți tabelul de adevăr al unui comparator de 2 biți.

❸ Câte intrări şi câte ieşiri poate avea un codificator? Câte intrări şi câte ieşiri poate avea un decodificator?

❹ CREEAZĂ! Pe panoul de comandă al imprimantei sunt montate butoanele *ON LINE* (funcționare sub controlul unității centrale), *OFF LINE* (funcționare autonomă), *LINE FEED* (avans de linie) şi *FORM FEED* (avans de pagină). Alcătuiți tabelul de adevăr al codificatorului care furnizează la ieşire următoarele combinații binare:

00 – *ON LINE;*

01 – *OFF LINE;*

10 – *LINE FEED;*

11 – *FORM FEED.*

❺ CREEAZĂ! Pe panoul de comandă al imprimantei sunt montate becurile (diodele luminiscente) indicatoare *READY* (disponibilă), *PAPER* (lipsa de hârtie), *TEST* (regimul de testare) şi *LOAD* (regimul de încărcare a informației). Alcătuiți tabelul de adevăr al decodificatorului care aprinde becurile în cauză. Stările respective sunt codificate prin următoarele combinații binare:

00 – *READY;*

01 – *PAPER;*

10 – *TEST;*

11 – *LOAD.*

❻ CREEAZĂ! Tastatura calculatorului include tastele funcționale <*F1*>, <*F2*>, ..., <*F12*>. Alcătuiți tabelul de adevăr al codificatorului care va furniza la ieşire numărul binar corespunzător tastei funcționale acționate.

❼ CREEAZĂ! Unitățile de intrare-ieşire ale unui calculator au următoarele adrese:

0000 – tastatura;

0001 – vizualizatorul;

0010 – imprimanta mecanică;

0011 – imprimanta cu jet de cerneală;

0100 – imprimanta laser;

0101 – unitatea de disc flexibil *A*;

0110 – unitatea de disc flexibil *B*;

0111 – unitatea de disc rigid *C*;

1000 – unitatea de disc rigid *D*.

Alcătuiți tabelul de adevăr al decodificatorului care va selecta unitatea indicată de adresa respectivă.

❽ CREEAZĂ! Operațiile aritmetice ale unui calculator ipotetic sunt codificate după cum urmează:

adunarea	– 000;
scăderea	– 001;
înmulțirea	– 010;
împărțirea	– 011;
compararea	– 100.

Alcătuiți tabelul de adevăr al decodificatorului operațiilor aritmetice.

❾ CREEAZĂ! Alcătuiți tabelul de adevăr al multiplexorului cu 2 linii de intrare.

5.5. Bistabilul *RS*

E cunoscut faptul că într-un circuit secvențial valorile variabilelor de ieşire depind nu numai de combinațiile valorilor variabilelor de intrare, dar şi de **consecutivitatea aplicării lor**. Cu alte cuvinte, circuitul secvențial memorizează informații referitoare la combinațiile aplicate la intrări în momentele precedente. Acest lucru este posibil datorită faptului că circuitele secvențiale sunt alcătuite din circuite combinaționale şi elemente de **memorie binară**.

Elementul de memorie binară este un circuit cu două stări distincte, destinat pentru a păstra o informație dintr-un singur bit. Circuitul respectiv are denumirea de bistabil.

În *figura 5.14* este prezentată schema circuitului bistabil de bază realizat cu porți logice *SAU-NU*, denumit **bistabil asincron *RS***.

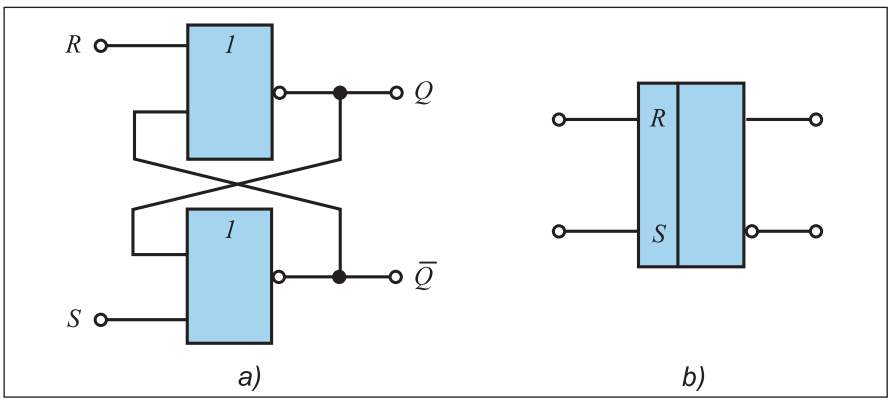


Fig. 5.14. Schema bistabilului asincron *RS* (*a*) şi simbolul utilizat (*b*)

Circuitul are două intrări notate cu R şi S şi două ieşiri notate cu Q şi $\overline{Q}$. Se observă că semnalele de ieşire Q şi $\overline{Q}$ sunt aplicate la intrările porților *SAU-NU*. Conexiunile respective sunt numite **reacții**. Tocmai datorită acestor conexiuni, circuitul dat are două stări distincte şi, prin urmare, asigură memorarea unui bit de informație.

Într-adevăr, admitem că semnalele de intrare $R = S = 0$, iar ieşirile $Q = 1$, $\overline{Q} = 0$. Datorită reacției, valoarea $Q = 1$ impune ieşirea $\overline{Q} = 0$. La rândul ei, tot datorită reacției, valoarea $\overline{Q} = 0$ confirmă ieşirea $\overline{Q} = 1$. Exact la fel, în cazul în care ieşirile $Q = 0$, $\overline{Q} = 1$, datorită reacției, valoarea $\overline{Q} = 1$ impune ieşirea $\overline{Q} = 0$. Prin urmare, valorile de intrare $R = S = 0$ nu schimbă starea bistabilului, acesta păstrând cifra binară memorată.

S-a convenit ca stării $Q = 1$, $\overline{Q} = 0$ să-i corespundă cifra binară 1, iar stării $Q = 0$, $\overline{Q} = 1$ cifra binară 0. Ieşirea Q este numită ieşire directă sau ieşire adevărată, iar ieşirea $\overline{Q}$ – ieşire inversă sau ieşire negată. Starea bistabilului este indicată de ieşirea directă Q.

Să examinăm cazul în care $R = 0$, $S = 1$. Presupunem că bistabilul se află în starea 0, adică $Q = 0$ şi $\overline{Q} = 1$. În acest caz, valoarea $S = 1$ va impune $\overline{Q} = 0$, care, la rândul ei, prin reacție, va asigura $Q = 1$. Prin urmare, bistabilul trece în starea 1 ($Q = 1$, $\overline{Q} = 0$). Această stare, după cum s-a stabilit anterior, se va păstra şi după trecerea semnalului S de la valoarea 1 la valoarea 0. Intrarea S, care asigură fixarea bistabilului în starea 1, poartă denumirea de **intrare de setare**.

În mod similar se poate constata că în cazul în care bistabilul se află în starea 1 ($Q = 1$, $\overline{Q} = 0$), iar $S = 0$, şi $R = 1$, bistabilul va trece în starea 0 ($Q = 0$, $\overline{Q} = 1$). Această stare se va păstra şi după trecerea semnalului R de la valoarea 1 la valoarea 0. Intrarea R, care asigură stabilirea bistabilului în starea 0, poartă denumirea de **intrare de resetare**.

Combinația de intrare $R = 1$ şi $S = 1$ impune valorile de ieşire $Q = 0$ şi $\overline{Q} = 0$. Valorilor de ieşire $Q = \overline{Q} = 0$ nu le corespunde nicio stare distinctă a bistabilului. Prin urmare, pentru bistabilul RS combinația de intrare $R = S = 1$ este o **combinație interzisă**.

Regimurile de funcționare a bistabilului în cauză sunt totalizate în *tabelul 5.1*.

Tabelul 5.1

Regimurile de funcționare a bistabilului asincron *RS*

Intrări		***Regim de funcționare***	***Ieşirea Q***
R	***S***		
0	0	păstrare bit	bit memorat
0	1	setare	1
1	0	resetare	0
1	1	interzis	–

Adjectivul *asincron* din denumirea bistabilului RS concretizează modul de influență a semnalelor de comandă R şi S asupra stării bistabilului. Din analiza circuitului prezentat în *figura 5.14* rezultă că semnalele de comandă, aplicate la intrările R şi S în momente arbitrare, pot schimba starea bistabilului.

Circuitele secvențiale în care starea circuitului poate fi schimbată de semnalele de comandă în momente arbitrare de timp se numesc circuite asincrone.

Un calculator modern conține zeci de mii de bistabile. Schimbarea stării lor în momente arbitrare, greu de controlat, poate fi cauza unor erori de funcționare. Pentru a evita o funcționare greșită, s-a convenit ca această comportare a circuitelor secvențiale să fie determinată în baza valorilor semnalelor de comandă aplicate la intrări în momente discrete, bine determinate în timp. Aceste momente sunt indicate cu ajutorul unor impulsuri speciale, numite **semnale de sincronizare**.

Circuitele secvențiale în care starea circuitului poate fi schimbată de semnalele de comandă numai în momentele indicate de semnalele de sincronizare se numesc circuite sincrone.

De obicei, semnalul de sincronizare se notează prin litera *C* (din engleză *Clock* „ceas") şi este furnizat de un dispozitiv special, numit **ceas de sistem**.

În *figura 5.15* este prezentată schema unui bistabil sincron denumit **bistabil sincron *RS***.

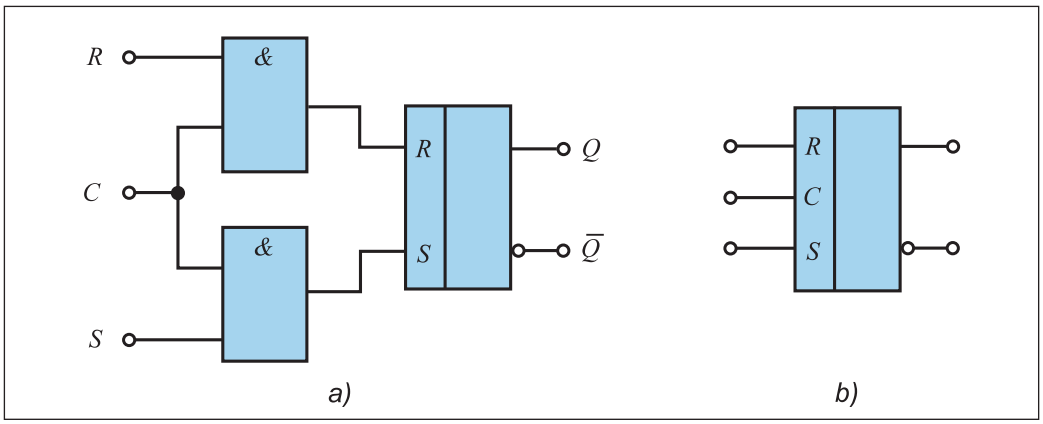


Fig. 5.15. Schema bistabilului sincron *RS* (*a*) şi simbolul utilizat (*b*)

Schema dată include bistabilul asincron *RS* (*fig. 5.14*) şi două porți logice *ŞI* care permit aplicarea semnalelor de comandă la intrările bistabilului asincron numai în cazul în care semnalul de sincronizare *C* are valoarea 1.

Întrebări şi exerciţii

❶ Prin ce se deosebesc circuitele combinaționale şi circuitele secvențiale?

❷ Care este destinația circuitului bistabil?

❸ Cum funcționează un circuit bistabil cu porți logice *SAU-NU*? Care este destinația reacțiilor?

❹ Explicați regimurile de funcționare a bistabilului asincron *RS*. De ce combinația $R = S = 1$ nu poate fi aplicată la intrările bistabilului examinat?

❺ STUDIU DE CAZ. Prin ce se deosebesc circuitele secvențiale asincrone şi circuitele secvențiale sincrone?

❻ Explicați cum funcționează bistabilul sincron *RS*. Care este destinația porților logice *ŞI* din componența acestui bistabil?

❼ În *figura 5.16* este prezentată schema circuitului bistabil de bază realizat cu porți logice *ŞI-NU*, denumit **bistabil asincron** $\overline{R}\overline{S}$.

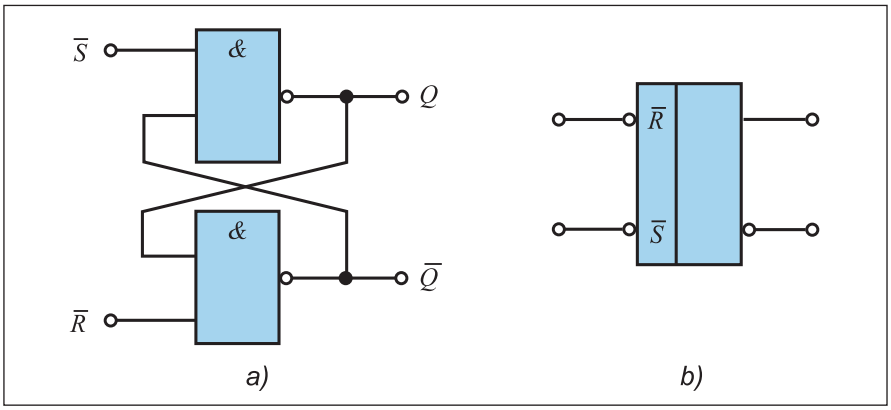


Fig. 5.16. Schema bistabilului asincron $\overline{R}\overline{S}$ (*a*) şi simbolul utilizat (*b*)

Circuitul are următoarele regimuri de funcționare:
– păstrare bit ($\overline{R} = 1$, $\overline{S} = 1$);
– setare ($\overline{R} = 1$, $\overline{S} = 0$);
– resetare ($\overline{R} = 0$, $\overline{S} = 1$).
Explicați cum funcționează bistabilul examinat. De ce combinația de intrare $\overline{R} = 0$, $\overline{S} = 0$ este o combinație interzisă?

❽ ELABOREAZĂ! Utilizând bistabilul asincron $\overline{R}\overline{S}$, elaborați schema **bistabilului sincron** $\overline{R}\overline{S}$.

❾ PROIECTEAZĂ! Desenați schema detaliată (la nivelul porților logice *ŞI*, *SAU-NU*) a bistabilului sincron *RS*. Pentru aceasta decalcați schemele respective din *figurile 5.14a* şi *5.15a*.

❿ PROIECTEAZĂ! Desenați schema detaliată (la nivelul porților logice *ŞI, ŞI-NU*) a bistabilului sincron $\overline{R}\overline{S}$.

5.6. Circuite secvenţiale frecvent utilizate

Registrul (*fig. 5.17a*, p. 158) este un dispozitiv numeric destinat păstrării temporare a unui număr binar,

$$D = d_{n-1} \ldots d_1 d_0.$$

Registrul este constituit din bistabile la care sunt ataşate circuite combinaționale ce permit înscrierea, citirea sau transferul informației. Înscrierea informației în registru se efectuează prin aplicarea la intrarea *W* (*Write* „înscrie”) a impulsului respectiv. Deoarece fiecare bistabil poate memora un singur bit, **capacitatea *n*** a unui registru este dată de numărul bistabilelor.

În anumite aplicații, de exemplu: înmulțirea şi împărțirea numerelor binare, scrierea sau citirea datelor pe disc, transmiterea datelor prin firele telefonice etc., apare necesitatea deplasării în stânga sau în dreapta a informației memorate într-un

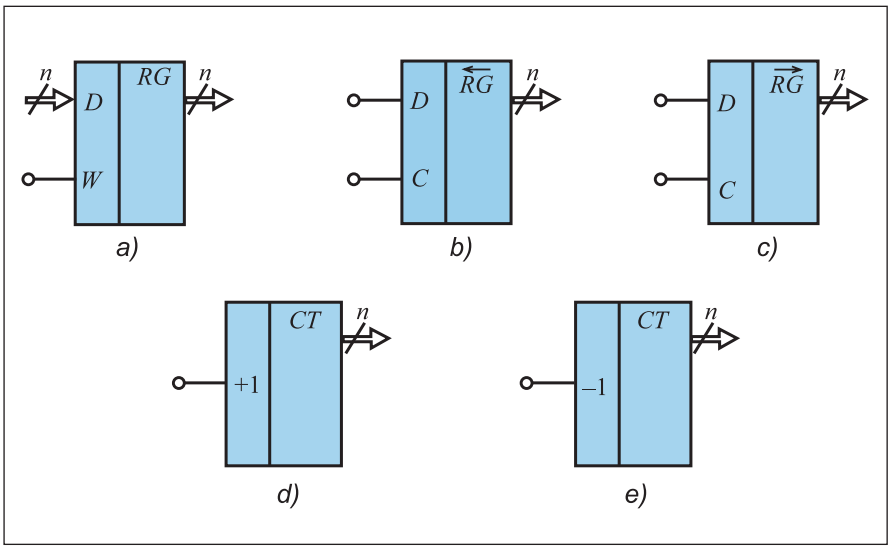


Fig. 5.17. Circuite secvenţiale frecvent utilizate:
a – registru; *b* – registru de deplasare de la dreapta spre stânga; *c* – registru de deplasare de la stînga spre dreapta; *d* – numărător direct; *e* – numărător invers

registru. În acest scop se utilizează **registrele de deplasare** (*fig. 5.17 b, c*).

Succesiunea stărilor unui **registru de deplasare de la dreapta spre stânga** este dată în *figura 5.18*.

Timpul	d_3	d_2	d_1	d_0
inițial	0	1	0	1
t_1	1	0	1	0
t_2	0	1	0	0
t_3	1	0	0	0
t_4	0	0	0	0
t_5	0	0	0	0
...	...			

Fig. 5.18. Succesiunea stărilor unui registru de deplasare de la dreapta spre stânga

Se consideră că starea inițială a registrului este 0101, iar impulsurile sunt aplicate la intrarea C în momentele de timp t_1, t_2, t_3 etc. Fireşte, registrul în cauză calculează produsul $D \times 2$.

În mod similar, un **registru de deplasare de la stânga spre dreapta** va calcula câtul $D : 2$.

Numărătoarele sunt circuite secvențiale care înregistrează numărul de impulsuri aplicat la intrare. Numărătoarele se clasifică după următoarele criterii:

– modul de codificare a informației (binar, binar-zecimal etc.);

– modul de modificare a stărilor numărătorului (numărătoare directe, inverse şi reversibile).

În general, numărătoarele sunt realizate prin asocierea circuitelor bistabile cu circuite combinaționale care determină modul corect în care numărătoarele urmează să-şi modifice starea la fiecare nou impuls sosit la intrare.

Un **numărător binar** înregistrează succesiunea impulsurilor aplicate la intrare în sistemul de numerație binar. **Capacitatea de numărare** a numărătorului binar depinde de numărul bistabilelor. Considerând pentru fiecare număr o stare distinctă, rezultă că acesta poate număra în gama 0 la 2^n–1, n fiind numărul bistabilelor.

În *figura 5.17d* este prezentat **numărătorul binar direct**, iar în *figura 5.19* tabelul succesiunii stărilor unui numărător binar direct de 3 biți.

Numărătoarele care îşi schimbă starea conform tabelului din *figura 5.19* sunt

Timpul	d_2	d_1	d_0
inițial	0	0	0
t_1	0	0	1
t_2	0	1	0
t_3	0	1	1
t_4	1	0	0
t_5	1	0	1
t_6	1	1	0
t_7	1	1	1
t_8	0	0	0
t_9	0	0	1
...		...	

Fig. 5.19. Succesiunea stărilor unui numărător binar direct de trei biţi

denumite **directe**, deoarece conținutul numărătorului creşte cu o unitate, la fiecare nou impuls sosit la intrarea +1. Dacă într-un numărător se introduce inițial un număr şi la fiecare impuls aplicat la intrarea –1 conținutul său scade cu o unitate, se obține un **numărător invers** (*fig. 5.17e*).

Întrebări şi exerciţii

❶ Care este destinația registrului? De ce depinde capacitatea unui registru?

❷ ANALIZEAZĂ! În registrul de deplasare de la dreapta spre stânga (*fig. 5.17b*) este încărcat numărul binar 1001. Care va fi conținutul registrului după aplicarea la intrarea C a unui impuls? Dar a două impulsuri?

❸ În registrul de deplasare de la stânga spre dreapta este încărcat unul dintre următoarele cuvinte binare:

a) 00000; *d)* 00100;

b) 10000; *e)* 00010;

c) 01000; *f)* 00001;

g) 10001; *i)* 01100;

h) 01010; *j)* 00110.

Care va fi conținutul registrului după aplicarea la intrarea *C* a două impulsuri consecutive?

❹ PROGRAMEAZĂ! Elaborați un program care simulează funcționarea unui registru de deplasare de la stânga spre dreapta de *n* biți.

❺ STUDIU DE CAZ. Care este destinația numărătoarelor? Cum se schimbă stările unui numărător direct? Dar ale unui numărător invers?

❻ ANALIZEAZĂ! Un numărător direct de 4 biți se află inițial în una dintre următoarele stări:

a) 0000; *f)* 1010;

b) 0010; *g)* 1100;

c) 0100; *h)* 1111;

d) 1000; *i)* 0101;

e) 1001; *j)* 0110.

Care va fi starea numărătorului după aplicarea a 5 impulsuri de intrare? Dar a 8 impulsuri?

❼ ANALIZEAZĂ! Un numărător invers de 4 biți se află în starea inițială 1001. Care va fi starea numărătorului după aplicarea a *m* impulsuri de intrare? Numărul *m* poate avea valorile 1, 4, 5, 8, 11, 17.

❽ PROGRAMEAZĂ! Elaborați un program care simulează funcționarea unui numărător direct de *n* biți.

5.7. Generatoare de impulsuri

Impulsurile se utilizează în echipamentele numerice pentru a asigura funcționarea secvențială a acestora. De obicei, generatoarele de impulsuri se realizează pe baza porților logice şi a elementelor de întârziere.

Elementul de întârziere reprezintă un circuit electronic care realizează funcția logică de repetare $y = x$, însă semnalul de ieşire y repetă semnalul de intrare x cu o întârziere de Δ unități de timp.

Din cursul de fizică e cunoscut faptul că viteza de propagare a semnalelor este finită. Prin urmare, orice conductor poate fi tratat ca un element de întârziere. Pentru a mări „inertitatea" circuitelor electronice şi pentru a realiza întârzieri semnificative, în componența lor se includ condensatoare şi rezistoare. În acest caz întârzierea Δ este determinată de capacitatea şi rezistența componentelor respective.

În *figura 5.20* sunt prezentate simbolul şi diagramele în timp ale elementului de întârziere.

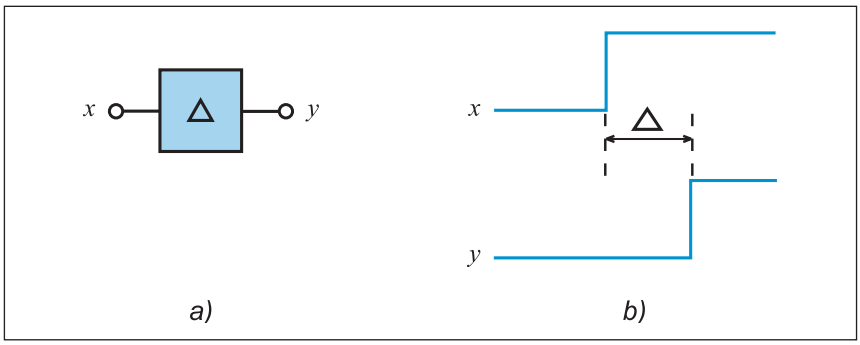


Fig. 5.20. Simbolul (*a*) şi diagramele în timp (*b*) ale elementului de întârziere

Schema unui **generator de impulsuri periodice** realizat pe baza unui element de întârziere şi a porţii logice *ŞI-NU* este prezentată în *figura 5.21*. În starea iniţială $x = 0$ şi $y = 1$, iar la ieşirea elementului de întârziere se menţine valoarea logică 1. Când la intrare se aplică semnalul de pornire $x = 1$, ieşirea devine egală cu 0 (*fig. 5.22*). În continuare, valoarea logică 0, după o întârziere Δ, este aplicată la a doua intrare a porţii logice *ŞI-NU*. Astfel, ieşirea y devine egală cu 1. Valoarea logică 1, după o întârziere Δ, va fi din nou aplicată la intrarea porţii logice *ŞI-NU*, impunând la ieşire valoarea $y = 0$ etc.

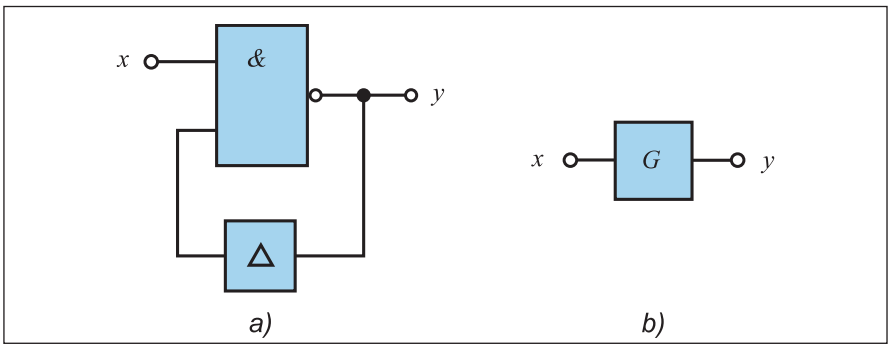


Fig. 5.21. Schema (*a*) şi simbolul generatorului de impulsuri periodice (*b*)

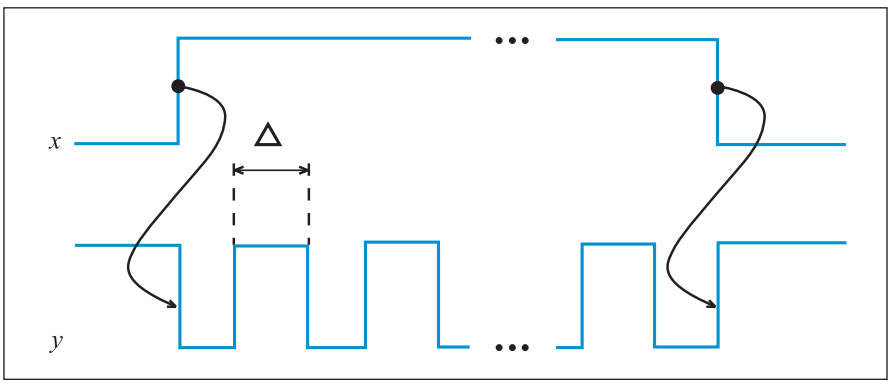


Fig. 5.22. Diagramele în timp ale generatorului de impulsuri periodice

Prin urmare, la ieşirea y a generatorului se va forma o succesiune de impulsuri cu durata Δ. Procesul de generare poate fi întrerupt prin aplicarea la intrare a semnalului de oprire $x = 0$.

Întrebări şi exerciţii

❶ Care este destinaţia elementului de întârziere? Desenaţi diagramele în timp ale elementului examinat.

❷ Explicaţi cum funcţionează generatorul de impulsuri periodice. De ce depinde durata impulsurilor?

❸ PROIECTEAZĂ! Înlocuiţi poarta logică *ŞI-NU* din componenţa generatorului de impulsuri periodice prezentat în *figura 5.21* printr-o poartă logică *SAU-NU*. Explicaţi cum va funcţiona circuitul respectiv. Desenaţi diagramele în timp ale generatorului obţinut.

❹ CERCETEAZĂ! E cunoscut faptul că variaţiile mărimilor fizice nu pot avea loc instantaneu. Prin urmare, orice poartă logică are o întârziere δ, denumită **întârziere parazitară**, valoarea concretă a căreia depinde de particularităţile circuitului respectiv.

Excludeţi din schema prezentată în *figura 5.21* elementul de întârziere, aplicând semnalul de la ieşirea porţii logice *ŞI-NU* direct la intrarea ei. Explicaţi cum va funcţiona circuitul obţinut. De ce depinde durata impulsurilor la ieşirea porţii logice? Desenaţi diagramele respective în timp.

❺ ANALIZEAZĂ! Circuitul secvenţial constă dintr-o poartă logică *NU,* semnalul de ieşire al căreia este aplicat direct la intrarea ei. Cum va funcţiona acest circuit?

Capitolul 6

STRUCTURA ŞI FUNCŢIONAREA CALCULATORULUI

6.1. Schema funcţională a calculatorului

Schema funcțională a calculatorului numeric este prezentată în *figura 6.1*.

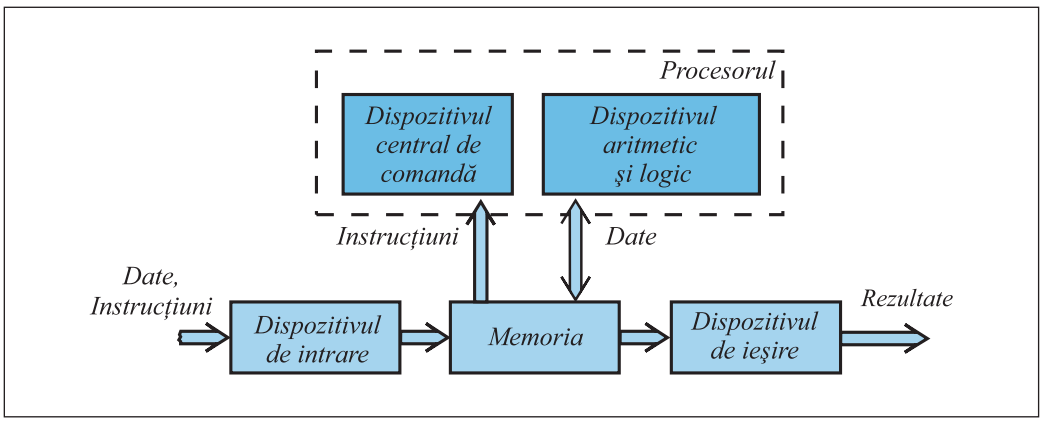


Fig. 6.1. Schema funcţională a calculatorului

Conform acestei scheme, calculatorul numeric conține următoarele **unități funcționale**:

– o unitate de memorie pentru a înmagazina datele inițiale, intermediare şi finale ale problemei, precum şi instrucțiunile care indică secvența calculelor;

– un dispozitiv aritmetic şi logic necesar efectuării operațiilor aritmetice şi logice elementare;

– unul sau mai multe dispozitive de intrare, respectiv, ieşire, necesare comunicării din exterior cu calculatorul;

– un dispozitiv central de comandă şi control care generează o succesiune de semnale de comandă necesare executării secvențiale a instrucțiunilor.

Dispozitivul aritmetic şi logic şi dispozitivul central de comandă formează **unitatea centrală de prelucrare a informației** sau, mai pe scurt, **procesorul**.

Memoria calculatoarelor moderne este organizată în două niveluri, şi anume: o unitate de memorie internă cu o viteză mare de lucru şi una sau mai multe unități de memorie externă cu o viteză mai redusă, însă cu o capacitate mult mai mare decât cea a memoriei interne.

Memoria internă (numită uneori memorie principală, centrală sau operativă) păstrează programul în curs de executare şi datele folosite de acesta, prezența ei fiind o condiție esențială pentru funcționarea calculatorului.

Memoria externă are rolul de a păstra cantități mari de informație şi programe folosite frecvent pentru a putea fi aduse într-un interval de timp mic în memoria internă. În prezent ca memorii externe sunt utilizate unitățile cu discuri sau benzi magnetice, unitățile cu discuri optice etc.

Unitățile de memorie externă şi dispozitivele de intrare-ieşire sunt numite **echipamente periferice**.

Pentru a asigura o interacțiune eficientă a procesorului, a memoriei interne şi a echipamentelor periferice, în cazul calculatoarelor personale schema funcțională se realizează fizic conform schemei-bloc prezentate în *figura 6.2*.

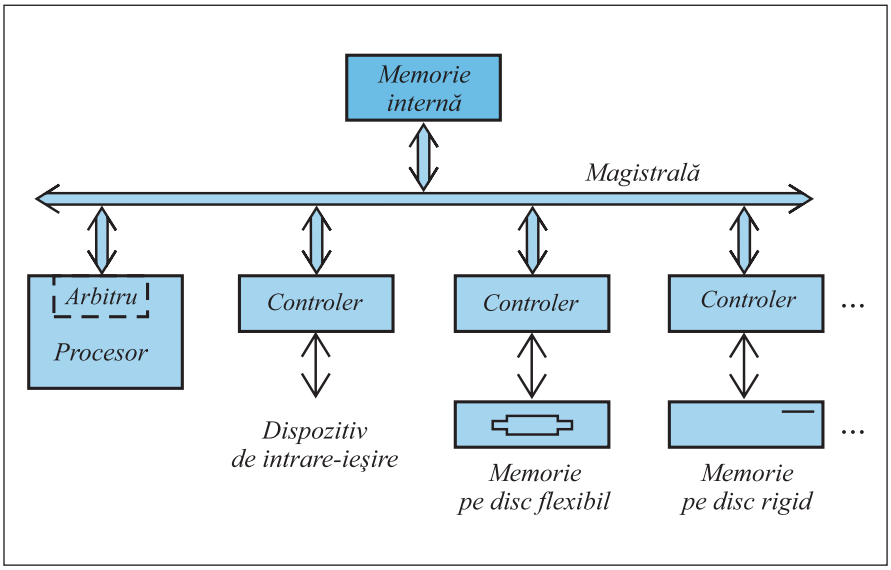


Fig. 6.2. Schema-bloc a unui calculator personal

Din analiza *figurii 6.2* rezultă că toate calculatoarele moderne au o **configurație modulară**. Fiecare modúl, şi anume controlerul, imprimantele, unitățile de disc magnetic etc., funcționează şi, în consecință, pot fi incluse sau excluse din componența calculatorului independent unul de altul. Prin urmare, configurația calculatorului poate fi modificată în funcție de destinația sistemului de calcul.

De exemplu, un sistem editorial va include mai multe tipuri de imprimante: mecanice pentru textele în curs de prelucrare, laser sau color pentru paginile machetate deja, cititoare de desene şi fotografii (scanere) etc. Un sistem destinat gestionării rapide a unui volum mare de date va include mai multe discuri magnetice, iar un calculator utilizat pentru montarea filmelor video va fi dotat cu camere de luat vederi şi vizualizatoare de o rezoluție adecvată, cu tastaturi similare pupitrului regizoral etc.

Întrebări şi exerciţii

❶ Numiți unitățile funcționale ale calculatorului şi explicați destinația lor.
❷ Care este rolul memoriei interne? Cum se realizează memoria externă a calculatoarelor moderne?
❸ Numiți echipamentele periferice pe care le cunoaşteți dvs.
❹ Numiți componentele unui calculator personal şi explicați destinația lor.
❺ Care este structura şi cum interacționează componentele unui calculator?
❻ Cum poate fi modificată configurația unui sistem de calcul? Care sunt avantajele configurației modulare a calculatorului?
❼ CERCETEAZĂ! Desenați schema-bloc a calculatorului cu care lucrați dvs. Care componente sunt obligatorii şi care opționale pentru funcționarea calculatorului?
❽ EXPERIMENTEAZĂ! Cum poate fi conectată la un calculator personal o unitate suplimentară de disc magnetic? O imprimantă laser? Un cititor de documente? O cameră de luat vederi?
❾ EXPLOREAZĂ! Utilizând un motor de căutare, colectați din Internet informațiile referitoare la evoluția schemelor funcționale ale calculatoarelor digitale şi scrieți un mic eseu despre acest subiect. Puneți în evidență factorii ce au dus la utilizarea pe scară largă a schemelor funcționale cu magistrală.

6.2. Formatul instrucţiunilor

Pentru a rezolva o problemă, calculatorul trebuie să cunoască în fiecare moment atât operația pe care urmează să o execute, cât şi datele care participă în operație. Aceste operații sunt comunicate calculatorului prin intermediul instrucțiunilor.

Instrucţiunea calculatorului reprezintă o succesiune de cifre binare prin care se indică procesorului operația de executat şi amplasamentul (locul) operanzilor.

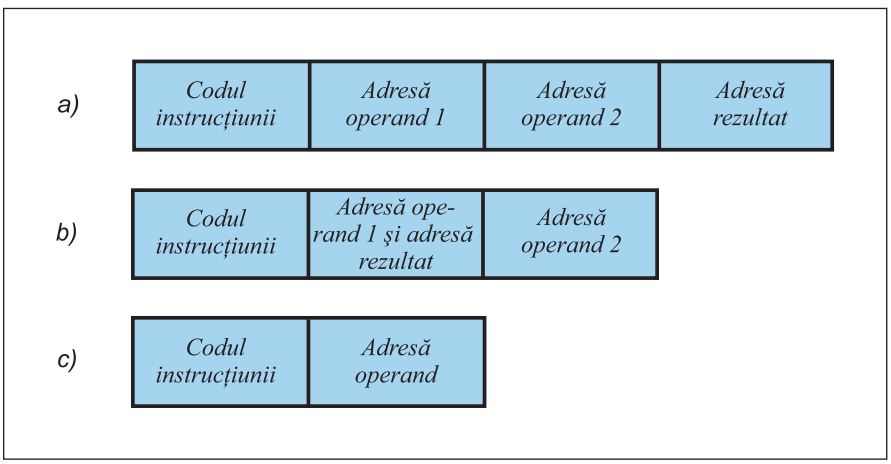


Fig. 6.3. Formatul instrucţiunilor cu trei (*a*), două (*b*) şi o singură adresă (*c*)

Succesiunea binară respectivă, denumită uneori şi **cuvânt instrucţiune**, este împărţită în câmpuri, fiecare câmp având o semnificaţie precisă. Numărul şi semnificaţia câmpurilor poartă denumirea de **formatul instrucţiunii**. În *figura 6.3*, p. 165 sunt prezentate formatele utilizate în calculatoarele moderne.

În general, pentru executarea unei operaţii este necesar ca instrucţiunea să conţină trei adrese (*fig. 6.3a*). Primele două adrese sunt folosite pentru obţinerea celor doi operanzi asupra cărora se va efectua operaţia specificată de câmpul *Codul instrucţiunii*. Rezultatul operaţiei va fi depus pe adresa specificată de câmpul *Adresă rezultat*.

Să analizăm un exemplu. Presupunem că operaţiile aritmetice şi logice sunt codificate după cum urmează (pentru simplitate vom utiliza echivalentele zecimale ale câmpurilor binare respective):

01 – adunarea;
02 – scăderea;
03 – operaţia logică *ŞI*;
04 – operaţia logică *SAU*.

Instrucţiunea

```
01 100 110 215
```

va impune procesorul să adune numerele din locaţiile 100 şi 110 şi să depună suma obţinută în locaţia 215.

Instrucţiunea

```
02 100 110 215
```

comunică procesorului că din numărul înscris în locaţia 100 se scade numărul din locaţia 110. Rezultatul obţinut va fi depus în locaţia 215.

În mod similar, instrucţiunea

```
03 200 300 100
```

specifică operaţia logică *ŞI* asupra biţilor cuvintelor din locaţiile 200 şi 300. Rezultatul operaţiei va fi depus în locaţia 100.

Se observă că într-o instrucţiune nu se specifică **valoarea operanzilor**, ci **adresele locaţiilor** în care pot fi găsiţi operanzii respectivi. Acest fapt permite utilizarea unuia şi aceluiaşi program pentru prelucrarea oricăror date iniţiale. Faptul că instrucţiunile lucrează cu adrese al căror conţinut trebuie prelucrat, şi nu cu însuşi conţinutul, constituie un principiu fundamental al calculatoarelor numerice, care permite ca un program să fie elaborat şi introdus în calculator independent de datele concrete asupra cărora se aplică.

În formatul cu trei adrese (*fig. 6.3a*), adresele sunt **specificate explicit**. Pentru o reprezentare mai compactă a instrucţiunilor se utilizează **specificarea implicită** a unor adrese. În acest caz, cuvântul instrucţiune nu conţine un câmp special pentru adresa implicită.

Dacă rezultatul obţinut în urma executării unei instrucţiuni se depune pe adresa unuia dintre operanzi, formatul respectiv va avea numai două adrese (*fig. 6.3b*). Prin

urmare, adresa rezultatului este specificată implicit. De exemplu, instrucțiunea

```
01 100 110
```

va impune procesorul să adune numerele din locațiile 100 şi 110 şi să depună suma obținută în locația 100. Evident, după înscrierea sumei, numărul inițial din locația 100 va fi suprimat.

S-a constatat că formatul cu două adrese, în prezent cel mai răspândit, permite scrierea de programe având un număr de instrucțiuni comparabil cu cel obținut atunci când s-ar utiliza mai multe adrese.

Formatul cu o singură adresă (*fig. 6.3c*) se utilizează în calculatoarele procesorul cărora include un registru special, denumit **acumulator**. În acumulator se păstrează primul operand şi se depune rezultatul executării operației respective. Prin urmare, adresa primului operand şi adresa rezultatului sunt specificate implicit. De exemplu, instrucțiunea cu o singură adresă

```
01 100
```

va aduna numărul din acumulator cu numărul din locația 100, iar suma obținută va fi depusă în acumulator. Respectiv, numărul inițial din acumulator va fi suprimat.

Instrucțiunile cu o singură adresă sunt eficiente din punctul de vedere al lungimii cuvântului şi al rapidității calculatorului. Totuşi un program scris cu instrucțiuni având o singură adresă va conține mai multe instrucțiuni decât în cazul în care se folosesc instrucțiuni cu două sau trei adrese.

Menționăm că toate calculatoarele moderne pot avea instrucțiuni de diferite formate. Informația referitoare la formatul fiecărei instrucțiuni se indică în câmpul *Codul instrucțiunii*.

Întrebări şi exerciţii

❶ Enumerați formatele instrucțiunilor utilizate în calculatoarele moderne. Explicați modul de specificare implicită a unor adrese.

❷ Explicați semnificația câmpurilor instrucțiunilor cu trei sau două adrese.

❸ Cum se specifică adresele operanzilor şi ale rezultatului în cazul instrucțiunilor cu o singură adresă?

❹ Explicați cum vor fi executate următoarele instrucțiuni cu trei adrese:

a) `01 200 201 202;` *c)* `03 100 150 250;`

b) `04 202 201 200;` *d)* `02 250 300 310.`

Operațiile aritmetice şi logice sunt codificate ca şi în exemplul din paragraful de față.

❺ Care va fi conținutul locației 100 după executarea instrucțiunii

```
01 200 300 100,
```

dacă în locațiile 200 şi 300 sunt înscrise numerele 17 şi, respectiv, 31?

❻ Explicați cum vor fi executate următoarele instrucțiuni cu două adrese:

a) 01 200 201; *c)* 03 100 150;

b) 04 202 201; *d)* 02 250 300.

❼ Care va fi conținutul locației 200 după executarea instrucțiunii

01 200 100,

dacă în locațiile 100 şi 200 sunt înscrise numerele 18 şi, respectiv, 32?

❽ Explicați cum vor fi executate următoarele instrucțiuni cu o singură adresă:

a) 01 100; *c)* 02 400;

b) 03 200; *d)* 04 150.

❾ Care va fi conținutul acumulatorului după executarea instrucțiunii

01 100,

dacă inițial în locația 100 era înscris numărul 12, iar în acumulator numărul 26?

❿ Care sunt avantajele şi dezavantajele formatelor cu trei, două sau cu o singură adresă?

⓫ STUDIU DE CAZ. Găseşte pe Internet informații referitoare la tipurile de calculatoare ce utilizează formate cu trei, două sau cu o singură adresă. Identificați domeniile de utilizare, avantajele şi neajunsurile tipurilor respective de calculatoare.

⓬ EXPLOREAZĂ! Utilizând un motor de căutare, colectați din Internet informațiile referitoare la evoluția formatelor instrucțiunilor utilizate în calculatoarele digitale şi scrieți un mic eseu despre acest subiect. Puneți în evidență factorii ce au dus la utilizarea pe scară largă a formatelor utilizate în calculatoarele personale.

6.3. Tipuri de instrucţiuni

Instrucțiunile unui calculator se împart în patru grupe:

– instrucțiuni operaționale, care efectuează operații aritmetice şi logice asupra datelor specificate prin operanzi;

– instrucțiuni de transfer, care deplasează informația între registre şi/sau locații fără a modifica informația transferată;

– instrucțiuni de salt, care în urma verificării unor condiții modifică analiza şi execuția secvențială a instrucțiunilor din program;

– instrucțiuni de intrare-ieşire care permit comunicarea calculatorului cu exteriorul.

Instrucțiunile operaționale prelucrează datele păstrate în locațiile memoriei interne şi în registrele procesorului. Cele mai cunoscute instrucțiuni ale acestei grupe sunt cele care efectuează operațiile aritmetice de bază: *adunarea*, *scăderea*, *înmulțirea* şi *împărțirea*.

Instrucțiunile logice de tipul *ŞI*, *SAU*, *NU* sunt instrucțiuni operaționale care acționează asupra pozițiilor individuale ale informației binare. În categoria instrucțiunilor operaționale întâlnim şi instrucțiuni de tipul: *şterge* conținutul unei locații sau al unui registru, *complementează* conținutul unei locații, *creşte* cu o unitate conținutul unui registru etc. În fine, în categoria instrucțiunilor operaționale sunt incluse instrucțiunile de *deplasare* a informației, în care partea de adresă a instrucțiunii conține un număr întreg, ce specifică numărul pozițiilor cu care se face deplasarea.

Instrucțiunile de transfer deplasează informația dintre locațiile memoriei interne, între registre sau între locații şi registre fără a altera conținutul informației transferate. Instrucțiunea trebuie să specifice explicit sau implicit adresa-sursă şi adresa de destinație a transferului. În timpul transferului şi după transfer, informația din sursă rămâne neschimbată. Cele mai uzuale instrucțiuni ale acestei grupe sunt cele prin care conținutul unei locații trece într-un anumit registru, registrul acumulator, precum şi instrucțiunea de transfer invers: dintr-un registru într-o locație a memoriei interne.

Instrucțiunile de salt se utilizează pentru modificarea ordinii de execuție a instrucțiunilor. În mod normal, instrucțiunile unui program sunt analizate şi executate în mod secvențial, în ordinea în care sunt aşezate în memorie. Această ordine poate fi schimbată cu ajutorul instrucțiunilor de salt condiționat sau necondiționat.

Instrucțiunile de salt condiționat permit alegerea continuării programului pe o anumită ramură, în funcție de o condiție de test realizată. Folosirea instrucțiunilor de salt condiționat dau posibilitate utilizatorului să introducă decizii logice în procesul execuției programului.

O instrucțiune de salt necondiționat conține, în partea de adresă, adresa instrucțiunii care va fi executată în continuare.

Instrucțiunile de intrare-ieşire permit comunicarea calculatorului cu echipamentele periferice. Echipamentul cu care se va efectua operația de intrare-ieşire se specifică în partea de adresă a instrucțiunii. De regulă, instrucțiunile de acest tip conțin atât informații legate de natura schimbului de date, adică introducerea sau extragerea lor, cât şi comenzi necesare funcționării corecte a periferiei. Tot în aceste instrucțiuni se specifică şi registrele sau locațiile în care vor fi depuse sau din care vor fi luate datele respective.

Întrebări şi exerciții

❶ Cum se clasifică instrucțiunile unui calculator? Care este destinația instrucțiunilor din fiecare grupă?

❷ Dați câteva exemple de instrucțiuni operaționale. Estimați numărul instrucțiunilor operaționale posibile.

❸ Care este destinația instrucțiunilor de transfer? Estimați numărul instrucțiunilor posibile de transfer.

❹ Când şi cum se utilizează instrucțiunile de salt? Care condiții de test pot fi analizate de aceste instrucțiuni?

❺ Care este destinația instrucțiunilor de intrare-ieşire? Ce informații conțin aceste instrucțiuni?

❻ EXPLOREAZĂ! Aflați clasificarea şi componența instrucțiunilor calculatoarelor cu care lucrați dvs.

6.4. Limbajul cod-calculator şi limbajul de asamblare

Pentru a rezolva o problemă, în memoria calculatorului trebuie să fie încărcate programul respectiv şi datele de prelucrat. Instrucțiunile programului şi datele de prelucrat se înmagazinează în memoria internă sub forma unor succesiuni de cifre binare pe care dispozitivul central de comandă le poate extrage şi interpreta.

Programele reprezentate în formă de succesiuni binare direct executabile de calculator se numesc programe în limbaj cod-calculator sau programe în limbaj maşină.

Pentru utilizator programul în cod-calculator poate fi prezentat în formă de şiruri de cifre binare sau, mai compact, de cifre octale, zecimale sau hexazecimale organizate pe locații ale memoriei.

Elaborarea programelor în limbaj cod-calculator este un lucru extenuant şi ineficient. Pentru a simplifica procesul de elaborare a programelor, s-a convenit ca instrucțiunile să fie scrise într-un limbaj simbolic, denumit **limbaj de asamblare**. În acest limbaj codurile instrucțiunilor se reprezintă printr-un grup de caractere astfel ales, încât să sugereze cât mai bine natura operației. Acest grup de caractere, de regulă trei, este cunoscut sub numele **mnemonica instrucțiunii**.

De exemplu, codurile instrucțiunilor calculatorului ipotetic din paragraful precedent pot fi notate simbolic conform *tabelului 6.1*.

Tabelul 6.1

Mnemonica instrucțiunilor

Cod instrucțiune	*Mnemonica*	*Semnificația*
01	INC	*Încarcă acumulatorul*
02	MEM	*Memorează acumulatorul*
03	ADU	*Adunare*
04	SCD	*Scădere*
05	SLT	*Salt necondiționat*
06	SLTC	*Salt condiționat*
07	STP	*Stop*

Adresele locațiilor memoriei interne pot fi specificate prin denumiri simbolice alese de utilizator, denumiri care sugerează semnificația conținutului locațiilor respective.

De exemplu, locația 185, în care se va păstra numărul x, poate fi notată prin X, locația 213 pentru numărul y se va nota prin Y, locația 200, în care va fi depusă

suma $x + y$, se va nota prin S. În limbajul de asamblare fragmentul de program pentru adunarea numerelor x şi y va avea forma:

```
INC X
ADU Y
MEM S.
```

În general, există o corespondență directă între scrierea instrucțiunii în limbajul de asamblare şi scrierea în limbajul cod-calculator, ceea ce face uşoară translatarea (traducerea) limbajului de asamblare în limbajul cod-calculator.

Translatarea constă în înlocuirea mnemonicii instrucțiunii şi a adreselor simbolice prin şirurile binare respective. Această înlocuire este făcută de un program special, denumit program de asamblare sau asamblor.

Limbajele cod-calculator şi de asamblare sunt **limbaje dependente de calculator**. Această dependență constă în faptul că formatul, codurile şi mnemonica instrucțiunilor exprimă structura internă a calculatorului. Programele elaborate în aceste limbaje sunt cele mai scurte şi rapide, însă procesul de programare necesită un mare volum de muncă. Simplificarea procesului de programare se asigură prin utilizarea limbajelor independente de calculator (*FORTRAN*, *BASIC*, *PASCAL*, *C* etc.), în care operațiile de prelucrare şi tipurile de date nu sunt legate de echipamentele calculatorului. Însă, cu regret, detaşarea utilizatorului de structura internă a calculatorului diminuează eficacitatea programelor respective.

Întrebări şi exerciţii

❶ Care este diferența dintre limbajul cod-calculator şi limbajul de asamblare?

❷ Cum se exprimă codurile instrucțiunilor şi adresele locațiilor într-un limbaj de asamblare?

❸ Care este destinația şi cum se realizează translatarea programelor scrise într-un limbaj de asamblare?

❹ PROGRAMEAZĂ! Se consideră că denumirea simbolică X semnifică locația 100, denumirea Y – locația 101, iar denumirea S – locația 102. Exprimați în limbajul de asamblare (*tab. 6.1*) următoarele programe:

a)
```
01 100
04 101
02 102
02 100
```

b)
```
01 100
02 100
03 100
```

c)
```
01 101
04 100
02 102
```

d)
```
01 101
03 101
03 101
03 101
02 102
```

e)
```
01 100
02 101
03 101
02 100
```

f)
```
01 100
02 102
01 101
02 100
```

Care va fi conținutul locațiilor 100, 101 şi 102 până şi după execuția fiecărui program?

❺ Se consideră că denumirile simbolice *X*, *Y* şi *S* specifică, respectiv, locațiile 100, 200 şi 300. Translatați următoarele programe scrise în limbajul de asamblare:

a)
```
INC X
SCD Y
MEM S
INC Y
```

b)
```
INC X
INC X
SLTC S
STP
```

c)
```
INC Y
ADU X
MEM S
STP
```

d)
```
INC Y
ADU Y
ADU Y
ADU Y
MEM S
```

e)
```
INC X
MEM S
INC Y
MEM X
INC X
```

f)
```
INC X
MEM S
INC Y
MEM X
INC S
```

Explicați cum va fi executat fiecare program.

❻ STUDIU DE CAZ. Care este diferența dintre limbajele dependente şi limbajele independente de calculator? Care sunt avantajele şi dezavantajele fiecărui limbaj?

❼ EXPLOREAZĂ! Utilizând Internetul în calitate de sursă de informare, alcătuiți o scurtă caracteristică a limbajelor cod-calculator şi a celor de asamblare ale calculatoarelor cu care lucrați: codurile şi instrucțiunile, clasificarea acestora.

6.5. Resursele tehnice şi resursele programate ale calculatorului

Numărul total al instrucțiunilor unui calculator depinde, în primul rând, de capacitatea lui. În cazul unui sistem mare de calcul, acest număr poate depăşi 1 000, pe când la calculatoarele foarte mici el nu este mai mare de 100. O anumită operație poate fi efectuată la unele calculatoare cu o singură instrucțiune, pe când aceeaşi operație, în alte calculatoare, pentru care nu există o instrucțiune specifică, este efectuată cu ajutorul unei succesiuni de alte instrucțiuni existente.

Operațiile efectuate intern de componentele electronice ale calculatorului sunt cunoscute ca **operații implementate prin echipamente**, pe când operațiile efectuate cu ajutorul unei secvențe de instrucțiuni sunt cunoscute ca **operații implementate prin program**. De exemplu, operația de extragere a rădăcinii pătrate într-un anumit calculator poate fi efectuată prin echipament electronic, în alt calculator – printr-un subprogram. Delimitarea între implementarea prin echipamente şi program depinde de calculator. În *figura 6.4* este prezentată o astfel de delimitare în cazul unui calculator cu posibilități medii.

Operațiile implementate prin echipamente se efectuează prin execuția unei singure instrucțiuni, pe când o operație implementată prin program necesită execuția mai multor instrucțiuni. Prin urmare, operațiile implementate prin echipamente se efectuează mai rapid, însă calculatorul respectiv este mai complex şi, în consecință, mai scump. Din contra, operațiile implementate prin program se efectuează mai lent, însă calculatoarele respective sunt mai simple şi, evident, mai ieftine.

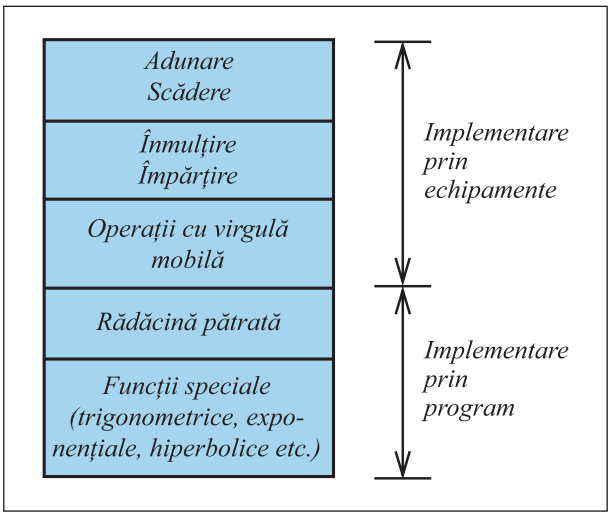


Fig. 6.4. Delimitarea între implementarea prin echipamente şi prin program

Din analiza principiilor de funcționare a procesorului rezultă că toate echipamentele unui calculator devin inutile în absența programelor care guvernează efectuarea operațiilor necesare pentru rezolvarea unei probleme. Exact în acelaşi mod, programele sunt fără niciun folos în absența echipamentelor numerice respective. Prin urmare, utilizarea tehnicii de calcul este posibilă numai în prezența atât a echipamentelor, denumite **resurse tehnice**, cât şi a programelor respective, denumite **resurse programate**.

Resursele tehnice ale unui sistem de calcul modern includ procesorul, memoria internă, unitățile de memorie externă, echipamentele de intrare-ieşire etc. **Resursele programate** vor include subprogramele care realizează operațiile implementate prin program, programele care simplifică accesul la unitățile de intrare-ieşire, asambloarele, editoarele de texte, compilatoarele limbajelor algoritmice şi, evident, programele elaborate de fiecare utilizator.

Menționăm că în literatura de specialitate resursele tehnice uneori sunt denumite prin cuvântul englez *hardware* („produse de metal”), iar resursele programate – prin cuvântul *software* („produse moi”). Respectiv, implementarea prin echipamente se numeşte implementare prin *hardware*, iar implementarea prin program – implementare prin *software*.

Întrebări şi exerciţii

❶ De ce depinde numărul total al instrucțiunilor unui calculator?

❷ Cum se efectuează operațiile în cazurile implementării prin echipamente şi în cele ale implementării prin program?

❸ EXPLOREAZĂ! Analizând repertoriul instrucțiunilor calculatorului la care lucrați dvs., determinați cum sunt implementate următoarele operații:
– înmulțirea şi împărțirea numerelor binare;
– adunarea şi scăderea numerelor cu virgulă mobilă;

– înmulțirea și împărțirea numerelor cu virgulă mobilă;
– extragerea rădăcinii pătrate;
– calcularea funcțiilor trigonometrice.

❹ STUDIU DE CAZ. Care sunt avantajele și dezavantajele implementării prin echipamente? Dar ale implementării prin program?

❺ EXPERIMENTEAZĂ! Numiți resursele tehnice și resursele programate ale unui sistem de calcul. Care sunt resursele respective în cazul calculatorului la care lucrați dvs.?

6.6. Memorii externe pe benzi și discuri magnetice

Principiul de funcționare a memoriilor examinate constă în înregistrarea informației pe un strat magnetic aflat în mișcare. Stratul magnetic este depus pe un suport neutru, acesta fiind, de regulă, o bandă de material plastic sau un disc de aluminiu. În calitate de strat magnetic se folosește mai frecvent oxidul de fier sau pelicule metalice subțiri de cobalt-nichel cu depunere în vid.

Înregistrarea sau citirea informației se efectuează cu ajutorul unui cap magnetic reprezentat în *figura 6.5*.

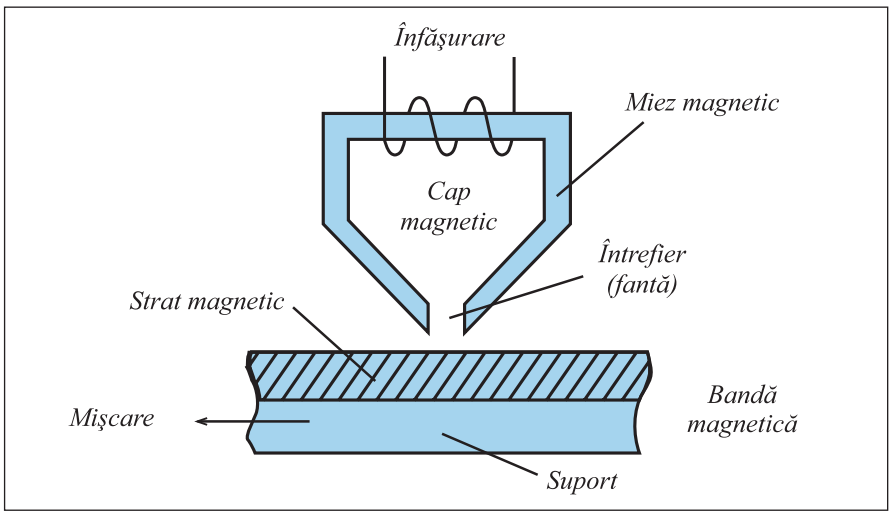


Fig. 6.5. Cap magnetic pentru înscrierea și citirea informației

Capul constă dintr-un miez, de regulă, din tole (foițe) de permalloy foarte subțiri (0,05 mm) și o înfășurare.

Într-un strat magnetic neînregistrat, câmpurile magnetice ale particulelor de oxid de fier sunt orientate haotic, anulându-se reciproc. Pentru **a înregistra** cifra binară 0 sau 1, prin înfășurarea capului magnetic se trece impulsul respectiv de curent. Impulsul care traversează înfășurarea creează în întrefier un câmp magnetic intens care magnetizează stratul aflat în momentul actual sub cap. Direcția de magnetizare, deci informația binară înscrisă, depinde de direcția

curentului în înfăşurarea capului magnetic. În *figura 6.6* este prezentat un exemplu de înregistrare a informației binare 101101 pe un strat magnetic aflat în mişcare.

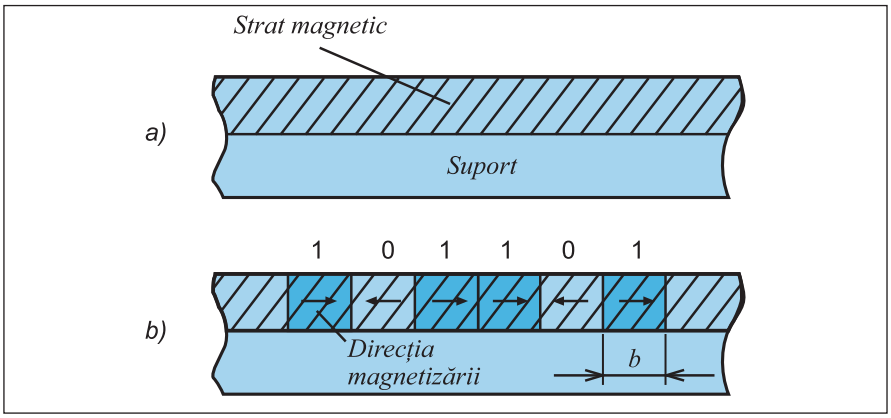


Fig. 6.6. Starea stratului magnetic până (*a*) şi după înregistrare (*b*)

Distanța *b* determină lungimea porțiunii necesare pentru a memora o cifră binară. Valoarea lui *b* depinde de viteza de mişcare a suportului, de proprietățile fizice ale stratului magnetic, de elementele constructive ale capului magnetic etc.

Numărul de elemente de memorie binară pe unitatea de lungime a suportului se numeşte densitatea de înregistrare a informației.

În cazul înregistrărilor magnetice, densitatea este dată de mărimea *1*/*b*. Valorile practice variază după tipul dispozitivului de memorare şi firma constructoare, fiind de ordinul sutelor şi miilor de biți pe milimetru de lungime a suportului.

În timpul operației de **citire** câmpul magnetic al particulelor de oxid de fier, trecând prin dreptul întrefierului (*fig. 6.5*), induce în înfăşurarea capului magnetic un semnal de ordinul 10^{-3} volți. Acest semnal este amplificat şi transformat în semnal-standard care reprezintă cifra binară respectivă 0 sau 1.

În marea lor majoritate **memoriile externe pe bandă magnetică** reprezintă echipamente periferice autonome care transferă informația spre/de la memoria internă după receptarea comenzilor corespunzătoare de la procesorul calculatorului. O unitate de memorie pe bandă magnetică (*fig. 6.7*, p. 176) include mecanismul de antrenare a benzii, dispozitivul de scriere-citire şi circuitele de comandă aferente.

O operație de citire sau scriere se realizează în timpul deplasării benzii. Între două operații succesive banda este oprită. Evident, informațiile înregistrate pot fi citite numai în ordinea amplasării lor fizice pe bandă. Datorită acestui fapt, unitățile de bandă magnetică sunt denumite unități de memorie externă cu **acces secvențial**.

Timpul necesar pentru selectarea unei informații din multitudinea datelor memorate pe un suport se numeşte timp de acces.

Timpul de acces al unității de bandă magnetică depinde de viteza benzii şi de locul amplasării informației de citit: la începutul, la sfârşitul sau la mijlocul benzii. În cazul informațiilor înregistrate la sfârşitul benzii, timpul de acces este de ordinul minutelor.

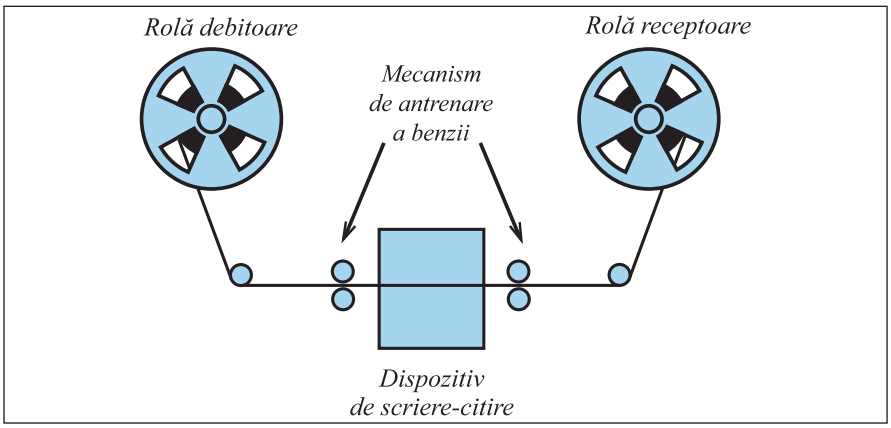


Fig. 6.7. Unitate de bandă magnetică

Capacitatea de memorare a unei benzi magnetice depinde de densitatea înregistrării, numărul de piste, lungimea benzii şi are valori de ordinul 10^8 octeți.

Din cauza timpului mare de acces şi capacitatea de memorie relativ mică, benzile magnetice se utilizează, în general, numai pentru arhivarea informației pe perioade mari de timp, până la 50 de ani.

Unitatea de discuri magnetice reprezintă la etapa actuală cea mai răspândită memorie externă a calculatoarelor numerice. Suportul informației este format dintr-un pachet de discuri, care poate fi fix sau amovibil, rotit cu o viteză de ordinul miilor de turații pe minut. Discurile sunt acoperite cu un strat de material feromagnetic. Informația este înregistrată pe ambele fețe de-a lungul unor piste concentrice. Pentru aceasta fiecare față este explorată în plan orizontal de un cap magnetic mobil (*fig. 6.8*).

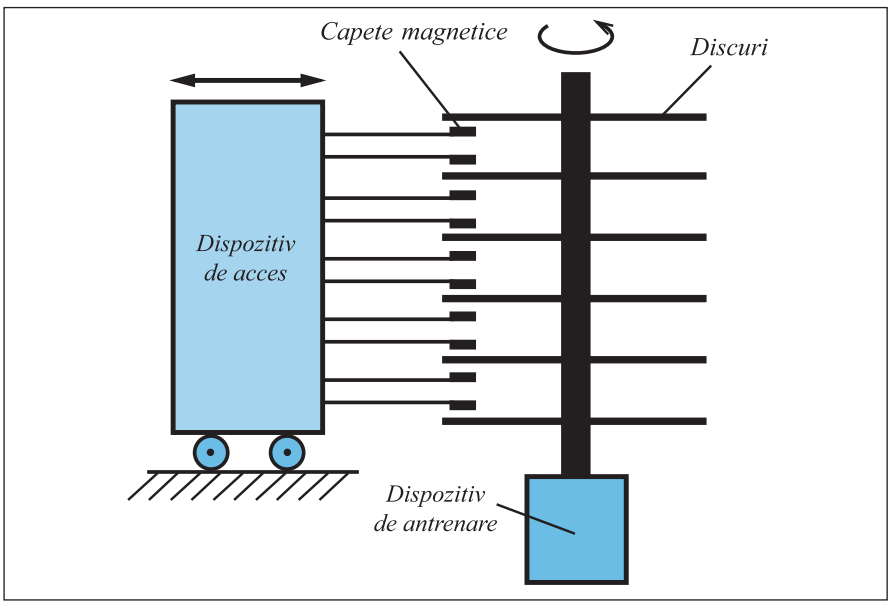


Fig. 6.8. Unitate de discuri cu capete mobile

Capetele sunt montate pe un braț extensibil şi acționate de un mecanism pentru a ajunge în dreptul pistei selectate.Toate capetele unei unități de memorie cu discuri sunt poziționate simultan.

Timpul de acces al unității de discuri se compune din timpul necesar deplasării ansamblului de capete magnetice de pe cilindrul curent pe cilindrul indicat şi din timpul necesar ca sectorul respectiv al discului să ajungă în dreptul capului magnetic. În practică se utilizează **timpul mediu de acces**, care pentru unitățile de disc moderne este de ordinul 10^{-3} secunde.

Amintim că în calculatoarele de performanță se utilizează unități de discuri cu capete magnetice fixe – câte un cap magnetic pentru fiecare pistă. Aceste unități asigură un timp de acces de ordinul 10^{-4} secunde, însă sunt foarte scumpe.

În trecut, pentru schimbul de informații între calculatoare se utilizau discuri singulare din material plastic, denumite **discuri flexibile** sau **dischete**. Aceste discuri erau încorporate într-o casetă din plastic sau în plicuri speciale. Organizarea fizică a informației era aceeaşi ca şi în cazul pachetelor de discuri, însă unitățile respective erau mult mai simple şi, evident, mai ieftine. Pentru a le deosebi de discurile flexibile, discurile convenționale ale calculatoarelor personale au fost denumite **discuri rigide**, **hard diskuri** sau **wincestere**.

Capacitatea de memorare a discurilor magnetice depinde de numărul de suprafețe ale pachetului, numărul de cilindri şi densitatea de înregistrare. La momentul actual este obținută capacitatea de ordinul 10^{12} octeți pentru un pachet de discuri.

Menționăm faptul că în ultimii ani, în locul sau suplimentar la discurile rigide au început să fie folosite dispozitive de stocare a datelor care foloseşte memorii cu semiconductori, denumite unități cu cipuri sau SSD-uri (Solid-State Drive). SSD-urile sunt mai rezistente la şocurile mecanice, având timpul de acces mai scăzut. La moment, capacitatea de memorare a acestor unități este mai mică decât a discurilor rigide, iar costul lor este mai mare.

Întrebări şi exerciţii

❶ Cum sunt reprezentate cifrele binare 0 şi 1 în înregistrările magnetice?

❷ Care este destinația capului magnetic?

❸ De ce depinde densitatea de înregistrare magnetică a informației?

❹ Cum se citeşte informația înregistrată pe un strat magnetic?

❺ Explicați cum funcționează unitatea de memorie pe bandă magnetică din *figura 6.7*.

❻ CERCETEAZĂ! De ce depinde capacitatea de memorare a unei benzi magnetice?

❼ O bandă magnetică are lungimea de 750 m. Înregistrarea informației se efectuează pe 8 piste plus pista bitului de paritate. Capacitatea de memorare a benzii este de 47 *Megaocteți*. Determinați densitatea de înregistrare.

❽ Viteza benzii magnetice este 2 m/s. În rola debitoare (*fig. 6.7*) sunt 750 m de bandă. Determinați timpul de acces la informația de la mijlocul benzii.

❾ Cum funcționează o unitate de discuri cu capete mobile?

❿ Cum este organizată informația pe un pachet de discuri magnetice?

⓫ Care este diferența dintre memoriile externe cu acces direct şi cu acces secvențial?

⓬ De ce depinde timpul de acces al unității de discuri magnetice?

⓭ CREEAZĂ! Utilizând sursele de informare din Internet, scrieți un mic eseu despre evoluția discurilor flexibile.

⓮ EXPERIMENTEAZĂ! Pentru unitatea de disc rigid cu care lucrați dvs. determinați:
– capacitatea discului;
– timpul mediu de acces.

⓯ CREEAZĂ! Utilizând sursele de informare din Internet, scrieți un mic eseu despre evoluția discurilor rigide.

⓰ STUDIU DE CAZ. Elaborați un mic studiu care pune în evidență avantajele şi neajunsurile discurilor rigide şi ale memoriilor cu semiconductori (SSD-urilor). Analizați frecvențele de utilizare a acestor unități de stocare a datelor în calculatoarele personale de tip desktop şi portabile (notebook-uri/laptop-uri) comercializate în țara noastră.

6.7. Memorii externe pe discuri optice

Principiul de funcționare a **memoriilor pe discuri optice** constă în înregistrarea informației pe un strat reflectorizant aflat în mişcare. Stratul reflectorizant din aluminiu, aur sau argint este depus pe un suport transparent din masă plastică.

În funcție de modul de scriere şi citire a informației, deosebim:

1) Discuri optice **numai pentru citire**. Informația pe astfel de discuri se înscrie de fabricant şi nu poate fi modificată de utilizator. Abrevierea engleză a acestor discuri este *CD-ROM* (***C**ompact **D**isc-**R**ead **O**nly **M**emory*).

2) Discuri optice **inscriptibile**. Informația pe astfel de discuri se înscrie de utilizator o singură dată, în continuare discul fiind disponibil numai pentru citire. Abrevierea engleză a acestor discuri este *CD-R* (***C**ompact **D**isc-**R**ecordable*).

3) Discuri optice **reinscriptibile**. Discurile în cauză permit mai multe cicluri de scriere/ştergere a informației. Abrevierea acestor discuri este *CD-RW* (***C**ompact **D**isc-**R**e**W**ritable*).

Pentru a asigura compatibilitatea unităților de citire, formatul datelor şi dimensiunile discurilor optice sunt standardizate. În *figura 6.9* este reprezentată structura discului optic *numai pentru citire* destinat publicului larg.

Înregistrarea cifrelor binare pe astfel de discuri constă dintr-o succesiune de adâncituri (în limba engleză *pit*) realizate pe una dintre suprafețele discului. Aceste adâncituri sunt despărțite de mici pauze şi sunt plasate pe suprafața discului sub forma unei piste în spirală.

Dimensiunile adâncițurilor sunt de ordinul unui micron (1 micron = 10^{-3} milimetri), distanța dintre spire este de 1,6 microni, lungimea spiralei fiind de 5 300 m. Discul conține 20 000 de piste (spire) pe care se află circa $6 \cdot 10^9$ adâncituri. Capacitatea de memorare a discului este de 640 *Megaocteți*.

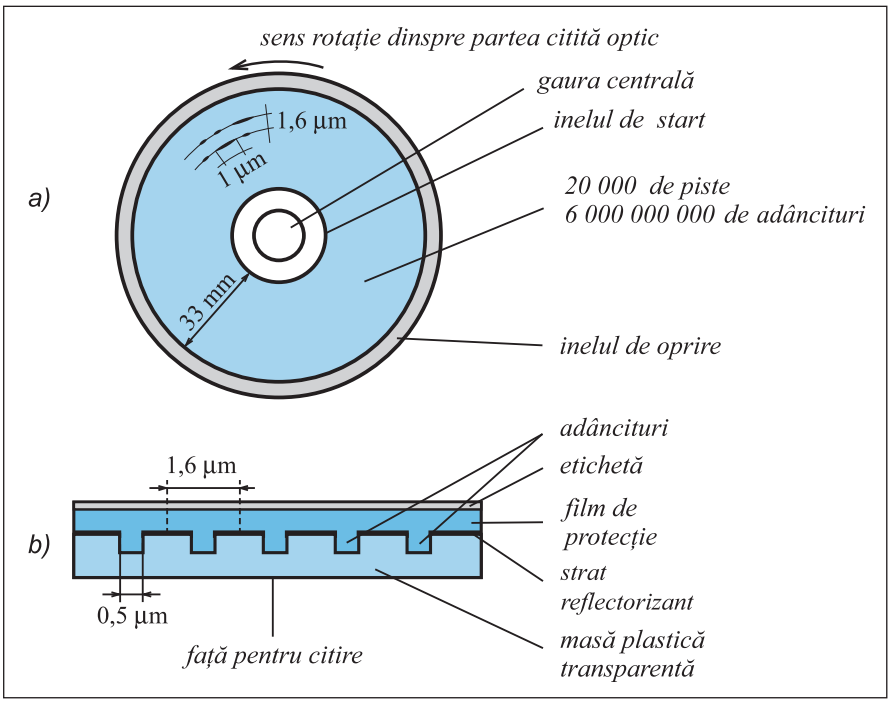


Fig. 6.9. Structura discului optic *CD-ROM*:
a – poziţionarea pistelor pe disc;
b – secţiune a discului perpendiculară pe piste

Citirea discului optic se realizează cu ajutorul unui fascicul de lumină care, după ce se reflectă de pe suprafața activă, este interceptat de o celulă fotosensibilă (*fig. 6.10*).

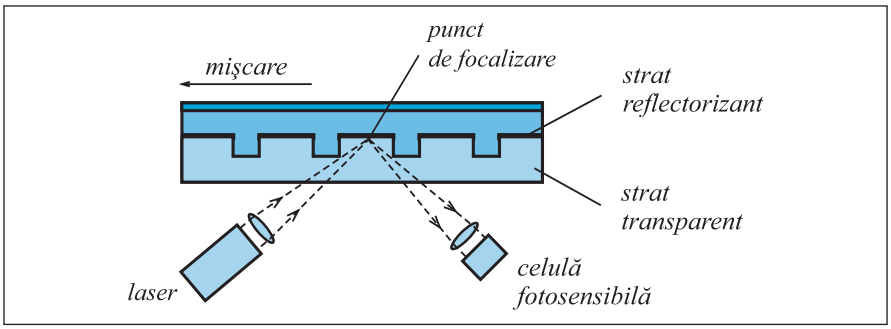


Fig. 6.10. Citirea discurilor optice

Parcurgând pistele respective, fasciculul laser este reflectat când stratul reflectorizant se află în **punctul de focalizare** şi nereflectat în caz contrar. Cu alte cuvinte, adâncimile de pe suprafața activă a discului optic schimbă (modulează) intensitatea fasciculului reflectat. În consecință, la ieşirea celulei fotosensibile se formează un semnal care redă succesiunea de cifre binare 0, 1 înregistrată pe disc la etapa fabricării.

În unitățile de disc optic moderne sursa de lumină – laserul şi celula fotosensibilă – fotodioda se realizează într-un dispozitiv integral, denumit **cap optic de citire**. Viteza unghiulară a discului este de 200–600 de turații pe minut, iar viteza liniară este de 1,4 m/s. În pofida vitezelor foarte mari, discul optic practic nu se uzează, întrucât între capul optic şi disc nu există un contact mecanic direct. Prin urmare, durata de exploatare a unui disc *CD-ROM* este determinată de calitatea stratului reflectorizant şi a stratului de protecție. În cazul utilizării aluminiului, din cauza oxidării, stratul reflectorizant se întunecă, reducând astfel viața unui disc optic la 10-15 ani. În cazul unui strat reflectorizant din aur, numai o acțiune violentă care ar distruge stratul protector ar putea afecta calitatea unui disc optic.

De regulă, discurile optice *CD-ROM* se utilizează pentru tirajarea sistemelor de operare, a mediilor de programare, a enciclopediilor, jocurilor electronice şi a altor informații destinate unui număr foarte mare de utilizatori.

Structura **discurilor optice inscriptibile** şi **reinscriptibile** este redată în *figura 6.11*.

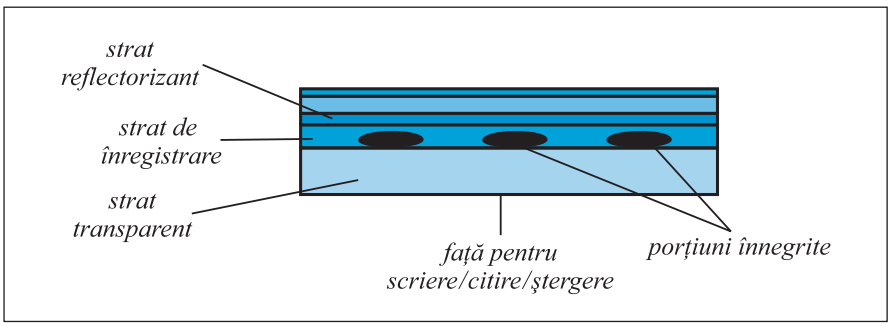


Fig. 6.11. Structura discurilor optice inscriptibile şi reinscriptibile

Din figură se observă că discurile examinate conțin un strat special, denumit **strat de înregistrare**. În cazul **discurilor inscriptibile**, acest strat este format dintr-un material organic – *cyanin* sau *phtalocyanin*, care se întunecă la încălzire. Înregistrarea informației se realizează cu ajutorul unui fascicul laser foarte puternic care înnegreşte porțiunile respective ale stratului de înregistrare.

La citire, porțiunile întunecate blochează trecerea fasciculului laser spre stratul reflectorizant, modificând astfel intensitatea luminii captate de celula fotosensibilă (*fig. 6.10*). Întrucât fasciculul laser folosit la citire este foarte slab, discul inscriptibil poate fi citit de mai multe ori fără a distruge informația înregistrată.

În cazul **discurilor reinscriptibile**, materialul din care este format stratul de înregistrare devine din nou transparent dacă este încălzit până la o temperatură specială, denumită **temperatură critică**. Evident, după ştergere, pe discul reinscriptibil pot fi înscrise date binare noi. Discurile moderne *CD-RW* suportă până la 10 000 de cicluri de scriere/ştergere.

Neajunsul comun al discurilor inscriptibile şi reinscriptibile este faptul că ele sunt mult mai sensibile la temperatură, durata lor de exploatare fiind mai mică decât a discurilor *CD-ROM*. Aceste discuri sunt şi mai scumpe, întrucât pentru o citire sigură stratul reflectorizant este realizat din aur sau argint.

De obicei, discurile optice *CD-R* şi *CD-RW* se utilizează pentru difuzarea informaţiilor operative unui cerc bine determinat de utilizatori – baze de date, pachete specializate de programe, rapoartele unor congrese importante, tablourile unei expoziţii, documente multimedia etc.

Menţionăm faptul că realizările tehnologice din ultimul deceniu au condus la apariţia discurilor optice de tipul *DVD* (*Digital Versatile Disc* – Disc Digital Multifuncţional). Capacitatea unui disc *DVD* este de aproape şapte ori mai mare decât a unui disc de tipul *CD*. Ulterior, au fost dezvoltate discuri şi mai noi, de tipul HVD (Holographic Versatile Disc – Disc Holografic Multifuncţional). Capacitatea unui astfel de disc este de mii de ori mai mare decât a unui disc de tipul CD.

În prezent, în calculatoarele personale de uz general, discurile optice şi, uneori, chiar şi cele rigide, sunt înlocuite cu memorii de tip *flash*. În aceste memorii informaţia este stocată cu ajutorul unor circuite electronice, ce asigură scrierea, citirea şi ştergerea datelor la viteze cu mult mai mari.

Întrebări şi exerciţii

❶ Cum sunt reprezentate cifrele binare 0, 1 în înregistrările optice?

❷ Care este capacitatea de memorare a unui disc optic?

❸ Cum se înregistrează informaţia pe un disc optic *CD-ROM*?

❹ De ce depinde capacitatea de memorare a unui disc optic?

❺ E cunoscut faptul că lungimea spiralei pe care se înregistrează informaţia unui disc optic este de 5300 m. Capacitatea de memorare a discului este de 640 *Megaocteţi*. Determinaţi densitatea de înregistrare a informaţiei pe discul optic.

❻ Care este domeniul de utilizare a discurilor *CD-ROM*?

❼ De ce depinde timpul de acces la informaţiile de pe un disc optic?

❽ Viteza liniară a discului optic este de 1,4 m/s. Cunoscând densitatea de înregistrare a informaţiei d = 128 *Kocteţi*/m, determinaţi **viteza de transmisie a datelor** de la unitatea de disc la unitatea centrală. Amintim că viteza de transmisie a datelor se măsoară în *biţi*, *Kbiţi* sau *Mbiţi* pe secundă.

❾ Pe cele 20 000 de piste (spire) ale unui disc optic sunt înregistraţi circa 640 de *Megaocteţi*. Câtă informaţie conţine o singură pistă a discului optic?

❿ Pe un disc *CD-ROM* sunt înregistrate în formă binară circa 74 de minute de muzică. Determinaţi câtă informaţie va conţine un cântec cu durata de 4 minute şi 30 de secunde. Câte piste va ocupa cântecul respectiv?

⓫ Cum se citeşte informaţia de pe un disc optic? Care este destinaţia capului optic de citire?

⓬ Explicaţi destinaţia straturilor unui disc optic inscriptibil. Cum se realizează înregistrarea informaţiei pe discul în cauză?

⓭ De ce depinde durata de exploatare a unui disc *CD-ROM*? Dar a discurilor *CD-R* şi *CD-RW*?

⓮ Cum se înregistrează şi se şterge informaţia de pe un disc optic reinscriptibil?

⓯ Care este domeniul de utilizare a discurilor *CD-R* şi *CD-RW*?

⓰ EXPERIMENTEAZĂ! Aflaţi parametrii tehnici ai unităţilor de discuri optice

cu care, posibil, erau/sunt dotate calculatoarele din laboratorul de informatică al liceului dvs.

⓱ CREEAZĂ! Utilizând sursele de informare din Internet, scrieți un mic eseu despre evoluția discurilor optice.

6.8. Vizualizatorul şi tastatura

Vizualizatorul este un dispozitiv de ieşire prin intermediul căruia informația este prezentată utilizatorului pe ecranul unui tub catodic. Schema funcțională a vizualizatorului este prezentată în *figura 6.12*.

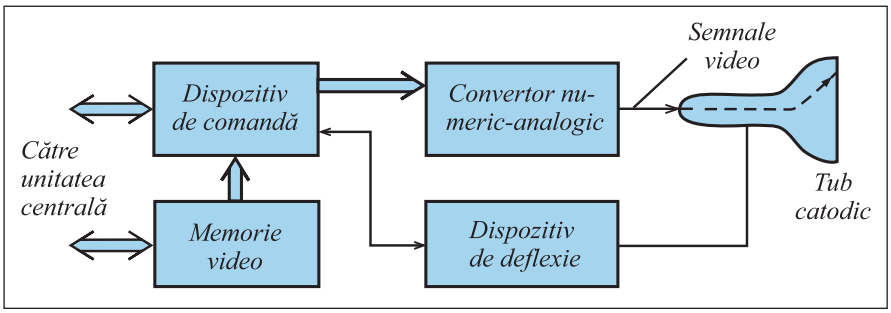


Fig. 6.12. Schema funcţională a vizualizatorului cu tub catodic

Ca şi în cazul televizoarelor pentru publicul larg, imaginea pe ecranul tubului catodic se formează din puncte. Punctele sunt activate de un fascicul de electroni. Culoarea şi luminanța fiecărui punct este controlată de semnale video, aplicate la intrările corespunzătoare ale tubului catodic. Parcurgerea punctelor într-o ordine prestabilită, de regulă, pe linii de la stânga la dreapta şi de sus în jos, este asigurată de dispozitivul de deflexie.

Fiecărui punct al ecranului îi corespunde o locație în memoria video a vizualizatorului. Locațiile conțin informații referitoare la culoarea şi luminanța punctelor respective. Dispozitivul de comandă citeşte locațiile memoriei video în ordinea parcurgerii punctelor de pe ecran. Conținutul fiecărei locații este interpretat drept o instrucțiune referitoare la modul de activare a punctului care îi corespunde. Semnalele video, necesare pentru comanda fasciculului de electroni, sunt formate de convertoare numeric-analogice.

Memoria video a vizualizatorului este încărcată de unitatea centrală a calculatorului. În orice moment, calculatorul poate modifica conținutul acestei memorii. În consecință, se va schimba şi imaginea afişată pe ecran. Imaginile pot fi animate prin modificarea conținutului memoriei video la frecvențe utilizate în cinematografie.

Vizualizatorul poate funcționa în unul dintre cele două regimuri: alfanumeric sau grafic.

În regim alfanumeric ecranul este împărțit în zone convenționale, numite **zone-caracter**. De regulă, aceste zone formează 25 de linii cu 80 de caractere pe linie. În fiecare zonă poate fi afişat un singur caracter dintr-un set de 256 de

caractere. Setul de caractere este constituit din literele mari şi mici ale alfabetului latin, cifrele zecimale, simbolurile matematice, semnele de punctuaţie şi unele caractere semigrafice, utilizate la afişarea pe ecran a tabelelor, diagramelor, chenarelor etc. Fiecare zonă-caracter poate avea o culoare pentru caracter şi altă culoare pentru fundal, ceea ce îi permite utilizatorului să afişeze pe ecran texte cu litere de diferite culori.

În regim grafic utilizatorul poate controla afişarea pe ecran a fiecărui punct. Numărul de puncte pe orizontală şi verticală determină rezoluţia vizualizatorului. De exemplu, expresia „rezoluţia 640×200" înseamnă că vizualizatorul are 640 de puncte pe orizontală şi 200 pe verticală.

Există mai multe norme internaţionale care reglementează rezoluţia şi numărul de culori ale vizualizatoarelor. Aceste caracteristici sunt recunoscute şi respectate de firmele producătoare.

De exemplu, în cazul calculatoarelor personale, cele mai răspândite sunt normele denumite *EGA*, *VGA* şi *SVGA*. Norma *EGA* stipulează că vizualizatorul are o rezoluţie de 640×350 de puncte, permiţând reprezentarea a 64 de culori.

Norma *VGA*, păstrând compatibilitatea cu *EGA*, oferă ca facilităţi suplimentare rezoluţia 640×480 de puncte şi 256 de culori distincte.

În scopul îmbunătăţirii calităţii imaginii redate, norma *SVGA* adaugă rezoluţia suplimentară de 1 024×768 de puncte.

În prezent, datorită realizărilor tehnologice din ultimii ani, ecranele cu tuburi catodice au fost înlocuite cu ecrane plate. Ecranul plat este alcătuit dintr-o matrice (un tablou) de celule emiţătoare de lumină, câte o celulă pentru fiecare din microzonele imaginii digitale furnizate de calculator. Utilizarea celulelor emiţătoare de lumină asigură o reducere substanţială a dimensiunilor vizualizatorului şi o îmbunătăţire semnificativă a rezoluţiei imaginilor afişate. Astfel de vizualizatoare se caracterizează printr-un consum redus de energie şi rezoluţii de până la 2 560 × 2 048 de puncte.

Tastatura este un dispozitiv de intrare care transformă acţionarea unei taste într-un cuvânt binar, accesibil echipamentelor calculatorului.

Partea electronică a unei tastaturi constă dintr-un codificator. La intrările codificatorului se aplică semnalele logice provenite de la taste. La ieşire se furnizează cuvintele unui cod binar, de obicei standard (*ISO*, *ACSII*, *UNICOD* etc.). Unele tipuri de tastaturi pot fi prevăzute cu un generator audio, care la acţionarea tastelor produce un sunet specific.

La noi, ca şi în ţările anglofone, cea mai răspândită este tastatura de tipul *QWERTY*, denumirea căreia provine de la amplasarea caracterelor *Q*, *W*, *E*, *R*, *T* şi *Y* în rândul de sus al tastelor alfanumerice. În ţările francofone se utilizează tastatura *AZERTY*, în Germania – *QWERTZ* etc.

Deşi locul amplasării tastelor şi numărul lor pot să difere, destinaţia tastelor este aceeaşi.

Tastatura dispune de următoarele grupe de taste: alfanumerice, funcţionale şi speciale. Grupul de taste alfanumerice include tastele cifrelor zecimale, tastele caracterelor alfabetului englez, tastele simbolurilor matematice şi ale semnelor de punctuaţie. Grupul de taste funcţionale include tastele <*F1*>, <*F2*>, ... ,<*F12*>.

Tastele respective nu au o destinație prestabilită şi semnificația lor este definită de programul care derulează pe calculator. Tastele speciale se utilizează pentru poziționarea cursorului vizualizatorului, pentru introducerea în calculator a unor cuvinte binare care nu au taste proprii etc.

În calculatoarele de performanță, vizualizatorul şi tastatura pot forma un echipament periferic unitar, denumit **consolă**. Consola utilizată de operator pentru dirijarea proceselor de calcul se numeşte **monitor**.

În cazul calculatoarelor de tip tabletă şi a celor încorporate, de exemplu, în telefoanele inteligente, echipamentele industriale cu dirijare numerică, electrocasnicele etc., în calitate de dispozitive de intrare-ieşire sunt utilizate **ecranele tactile** (*touchscreen*), care permit atât afişarea informațiilor vizuale, cât şi introducerea comenzilor şi datelor prin glisări şi atingeri a elementelor de control afişate pe ecran.

Întrebări şi exerciţii

❶ Explicați cum funcționează vizualizatorul. Care este destinația părților componente ale unui vizualizator?

❷ Cum poate fi schimbată imaginea afişată pe ecranul unui vizualizator? Cum pot fi animate imaginile de pe ecran?

❸ Prin ce se deosebesc regimurile de funcționare a vizualizatorului?

❹ Care sunt indicii principali de calitate ai unui vizualizator?

❺ EXPERIMENTEAZĂ! Determinați tipul vizualizatorului la care lucrați dvs. Aflați rezoluția şi numărul de culori disponibile.

❻ Care sunt părțile componente ale unei tastaturi? Cum se determină tipul tastaturii?

❼ Numiți grupurile de taste şi destinația lor.

❽ CERCETEAZĂ! Utilizând sursele de informație din Internet, elaborați un studiu comparativ al vizualizatoarelor cu tub catodic şi al celor cu ecran plat, punând în evidență atât avantajele, cât şi neajunsurile fiecăruia din ele. Indicați domeniile de utilizare a fiecărui tip de vizualizator.

❾ STUDIU DE CAZ. Cu ajutorul unui motor de căutare, găsiți pe Internet descrierile monitoarelor propuse de unitățile de comerț. Aflați cum parametrii tehnici ai monitoarelor influențează costurile, care sunt avantajele, neajunsurile şi domeniile de utilizare a acestora.

❿ CREEAZĂ! Utilizând Internetul în calitate de sursă de informație, scrieți un mic eseu despre evoluția vizualizatoarelor, atât a celor industriale, cât şi a celor de uz personal.

6.9. Imprimantele

Imprimantele sunt dispozitive de ieşire care furnizează rezultatele sub forma unui document tipărit. În funcție de **tehnica de tipărire** utilizată, imprimantele se pot clasifica în:

– imprimante mecanice, în care tipărirea se face prin acționarea unor ciocănaşe sau ace;

– imprimante laser, în care tipărirea se face folosind metode electrostatice ca la maşinile de copiat;

– imprimante cu jet de cerneală;

– imprimante termice, funcționarea cărora se bazează pe utilizarea unei hârtii speciale care îşi schimbă culoarea în funcție de temperatură.

Principiul de funcționare a **imprimantei matriciale cu ace** este prezentat în *figura 6.13*.

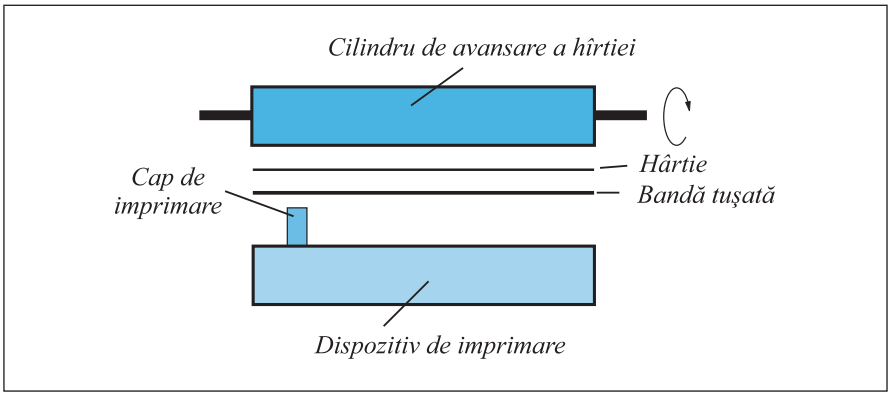


Fig. 6.13. Principiul de funcţionare a imprimantei matriciale cu ace

Capul de imprimare conține un grup de ace metalice subțiri care la momentul potrivit lovesc prin bandă tuşată în foaia de hârtie. Configurația ácelor ce lovesc în fiecare moment şi avansarea capului de-a lungul liniei de imprimat determină imaginile tipărite.

Imprimantele matriciale pot funcționa în regim grafic şi regim alfanumeric.

În **regim grafic,** calculatorul comandă tipărirea pe hârtie a fiecărui punct aparte. Evident, din puncte pot fi formate orice imagini: grafice, desene tehnice, caractere cu configurații concepute de utilizator. Imprimarea informației în regim grafic este lentă din cauza deplasărilor multiple ale capului de imprimare şi a caracterului discontinuu al avansului de hârtie cu paşi foarte mici.

În **regim alfanumeric**, calculatorul transmite imprimantei numai codurile caracterelor de tipărit. Fiecărui cod îi corespunde o imagine din puncte care se păstrează într-o memorie specială a imprimantei. Imaginile respective sunt tipărite prin una sau, cel mult, două-trei treceri ale capului de imprimare.

Imprimarea în regim alfanumeric este mult mai rapidă, însă pot fi tipărite numai caracterele imaginile cărora au fost, în prealabil, înmagazinate în memoria imprimantei. Imprimantele matriciale simple au câteva seturi de caractere, înscrise într-o memorie permanentă. Imprimantele matriciale mai performante permit încărcarea prin program a mai multe seturi de caractere cu configurații concepute de utilizator.

E cazul să subliniem că cu cât numărul ácelor din capul de imprimare este mai mare, cu atât calitatea imprimării este mai bună. În prezent se utilizează imprimante cu 9 sau cu 24 de ace. Calitatea imprimării poate fi îmbunătățită prin tipărirea repetată (de 2-4 ori) a aceluiaşi caracter pe acelaşi loc. Viteza de tipărire a imprimantelor mecanice cu ace este de 150-500 de caractere pe minut.

Principiul de funcționare a **imprimantelor laser** este prezentat în *figura 6.14*.

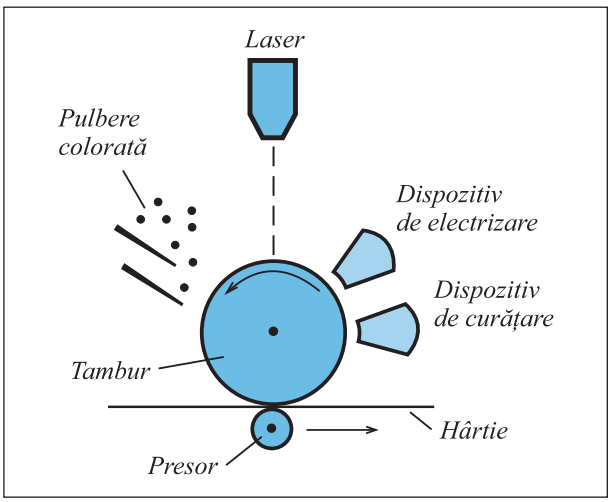


Fig. 6.14. Principiul de funcţionare a imprimantei laser

Elementul principal al acestei imprimante este un tambur acoperit cu un strat semiconductor, care își poate schimba proprietățile electrice sub acțiunea luminii. Mai întâi, suprafața tamburului se electrizează. Cu ajutorul razelor laser pe suprafața electrizată a tamburului se proiectează punctele imaginii de tipărit. Întrucât sectoarele care au fost luminate își schimbă conductibilitatea, sarcinile electrice respective se neutralizează. Prin urmare, pe suprafața tamburului se formează o imagine electrică ascunsă. Developarea imaginii se efectuează cu ajutorul unor particule de pulbere colorată, atrase de sectoarele electrizate ale tamburului. Imaginile de pe tambur se transferă pe hârtie şi se fixează prin topirea particulelor de pulbere.

În continuare, toate sarcinile electrice de pe suprafața tamburului sunt neutralizate, iar resturile de pulbere sunt curățate.

Imprimantele laser sunt cele mai bune dintre imprimantele moderne. Textele şi grafica tipărite la aceste imprimante nu se deosebesc de cele tipografice. Viteza de tipărire este de 5-15 pagini pe minut.

Imprimantele cu jet de cerneală formează punctele unei imagini din picături microscopice pulverizate pe hârtie de nişte ajutaje speciale. Ele asigură o calitate foarte bună a tiparului şi sunt utilizate pentru imprimarea color. Aceste imprimante sunt mai scumpe decât imprimantele cu ace şi necesită o îngrijire tehnică deosebită.

Imprimantele termice utilizează o matrice de ace a căror încălzire selectivă este determinată în funcție de caracterul ce urmează a fi tipărit. Viteza de imprimare este relativ mică, circa 300 de rânduri pe minut, însă avantajul principal constă în dimensiunile reduse ale imprimantei.

Întrebări şi exerciţii

❶ Cum se clasifică imprimantele în funcție de tehnica de tipărire?

❷ Care sunt părțile componente ale unei imprimante?

❸ Care este principiul de funcționare a imprimantei matriciale cu ace? Cum funcționează această imprimantă?

❹ Explicați cum funcționează o imprimantă laser. Care este avantajul principal al acestei imprimante?

❺ EXPERIMENTEAZĂ! Determinați tipul imprimantei cu care lucrați dvs. Aflați parametrii tehnici ai imprimantei: setul de caractere, regimurile de funcționare, capacitatea memoriei-tampon, viteza de imprimare.

❻ STUDIU DE CAZ. Cu ajutorul unui motor de căutare, găsiți pe Internet descrierile imprimantelor propuse de unitățile de comerț. Aflați cum parametrii tehnici ai imprimantelor influențează costurile, care sunt avantajele, neajunsurile şi domeniile de utilizare a acestora.

❼ CREEAZĂ! Utilizând Internetul în calitate de sursă de informație, scrieți un mic eseu despre evoluția imprimantelor, atât a celor industriale, cât şi a celor de uz personal.

6.10. Clasificarea calculatoarelor

Caracteristica generală a unui calculator include următoarele date:
– viteza de operare;
– capacitatea memoriei interne;
– componența, capacitatea şi timpul de acces ale unităților de memorie externă;
– componența şi parametrii tehnici respectivi ai echipamentelor periferice;
– parametrii de masă şi gabarit;
– costul.
În funcție de aceste date, calculatoarele moderne se clasifică în 4 categorii:
– supercalculatoare;
– calculatoare mari (macrocalculatoare);
– minicalculatoare;
– microcalculatoare (calculatoare personale).

Supercalculatoarele pot executa peste 10^{15} (1000 bilioane) de operații pe secundă, iar prețul lor depăşeşte 20 milioane de dolari. Cercetări şi proiectări în industria supercalculatoarelor se realizează în SUA şi Japonia de firmele *IBM*, *Gray Research*, *Fujitsu ETA Systems*, *Dell*, *Sutherland* etc. Supercalculatoarele se utilizează în prelucrări extrem de complexe ale datelor în aeronautică, fizica nucleară, astronautică, seismologie, prognoza meteo etc.

Calculatoarele mari (în engleză *mainframe* „dulapul principal") pot executa sute de bilioane de operații pe secundă (1 bilion = 10^{12}), prețul variind între 20 de mii şi câteva milioane de dolari. Schema-bloc a calculatoarelor date este prezentată în *figura 6.2*. De regulă, calculatoarele mari includ zeci de unități de disc magnetic şi imprimante, sute de console aflate la diferite distanțe de unitatea centrală. Aceste calculatoare se utilizează în cadrul unor mari centre de calcul şi funcționează în regim non-stop. Principalele firme producătoare de calculatoare mari sunt *IBM*, *Hitachi, Amdahl, Fujitsu* etc.

Minicalculatoarele efectuau sute de milioane de operații pe secundă, iar prețul lor nu depăşea 200-300 mii de dolari. Echipamentele periferice ale unui minicalculator includeau câteva discuri magnetice, una sau două imprimante, mai multe console. Minicalculatoarele erau mai uşor de utilizat şi operat decât calculatoarele mari şi se utilizau în proiectarea asistată de calculator, în automatizări industriale, pentru prelucrarea datelor în experimentele ştiințifice etc. Dintre firmele ce produceau minicalculatoare vom remarca *IBM*, *Wang*, *Texas Instruments*, *Data General*, *DEC, Hewlett-Packard* etc. În prezent minicalculatoarele au fost înlocuite de calculatoarele personale.

Microcalculatoarele, denumite şi calculatoare personale, sunt realizate la prețuri scăzute – între 100 şi 15 000 de dolari şi asigură o viteză de calcul de ordinul miliardelor de operații pe secundă. Schema-bloc a unui calculator personal este prezentată în *figura 6.2*.

De obicei, echipamentele periferice ale unui microcalculator includ una sau mai multe unități de disc rigid sau SSD (*Solid-State Drive*), o unitate de disc optic, o imprimantă şi o consolă. Structura modulară şi gruparea tuturor echipamentelor în jurul unei magistrale permite configurarea microcalculatorului în funcție de necesitățile individuale ale fiecărui utilizator.

În acest scop, microcalculatoarele dispun de câteva porturi USB (*Universal Serial Bus* – Magistrală Serială Universală), care permit conectarea diferitor echipamente periferice suplimentare, inclusiv a memoriilor de tip *flash*. De asemenea, calculatoarele personale, destinate plasării pe masa de lucru (de tip *desktop*), calculatoarele portabile (denumite *notebook*-uri sau *laptop*-uri) şi cele de tip tabletă, sunt dotate cu dispozitive periferice ce asigură comunicarea cu alte sisteme tehnice digitale prin tehnologii radio (*Wi-Fi*, *Bluetooth*) sau raze infraroşii.

Corporații care produc microcalculatoare există în foarte multe țări, însă lideri mondiali, unanim recunoscuți, sunt firmele *Lenovo*, *Apple*, *Hewlett-Packard*, *Dell*, *Asus*, *Acer*, *Samsung* etc.

Întrebări şi exerciţii

❶ EXPERIMENTEAZĂ! Numiți parametrii mai semnificativi ai unui calculator. Determinați parametrii respectivi ai calculatorului cu care lucrați dvs.

❷ Cum se clasifică calculatoarele în funcție de parametrii tehnici şi economici?

❸ CREEAZĂ! În baza informațiilor colectate din Internet, dați o caracteristică succintă a fiecărei categorii de calculatoare: supercalculatoare, calculatoare mari, minicalculatoare şi microcalculatoare. Puneți în evidență domeniile de utilizare a calculatoarelor din fiecare categorie.

❹ EXPLOREAZĂ! Folosind un server de căutare, găsiți pe Internet informații detaliate despre principalii producători de calculatoare personale. Alcătuiți un clasament al acestor producători după cota de piață deținută de ei.

❺ ANALIZEAZĂ! Identificați cota procentuală a profitului marilor companii ce produc calculatoare, pe categorii (supercalculatoare, calculatoare mari, macrocalculatoare, minicalculatoare, microcalculatoare), şi elaborați un mic studiu ce ar pune în evidență evoluția acestui sector important al economiei globale.

❻ CERCETEAZĂ! Utilizând datele publicate de Biroul Național de Statistică şi Agenția Națională pentru Reglementare în Comunicații Electronice şi Tehnologia Informației (ANRCETI), elaborați un mic studiu despre producerea şi comercializarea calculatoarelor în țara noastră.

❼ EXPLOREAZĂ! Un grup internațional de specialişti în domeniul informaticii implementează aşa-numitul proiect „Top 500”. Fiind lansat în 1993, acest proiect publică de două ori pe an lista primelor 500 celor mai puternice supercalculatoare. Elaborați un mic studiu despre primele zece supercalculatoare, punând în evidență parametrii tehnici şi destinația acestora, companiile care le produc şi rolul cooperării internaționale în dezvoltarea ştiinței calculatoarelor şi tehnologiilor informației.

6.11. Microprocesorul

Microprocesorul este un circuit integrat care implementează funcțiile unității centrale de prelucrare a informației, şi anume – extragerea şi executarea instrucțiunilor.

De regulă, un microprocesor conține un dispozitiv aritmetic şi altul de comandă, un grup de registre destinate păstrării temporale a datelor frecvent utilizate, magistrale şi circuite de comandă aferente (*fig. 6.15*).

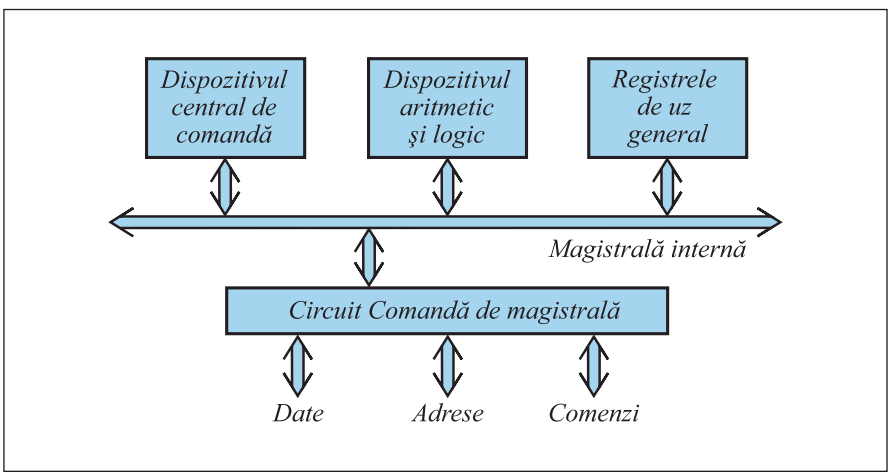


Fig. 6.15. Schema funcţională a unui microprocesor

Microprocesorul interacționează cu unitățile de memorie şi echipamentele periferice prin intermediul a trei magistrale: *Date*, *Adrese* şi *Comenzi*. Traficul de informații prin magistrale este controlat de circuitul *Comandă de magistrală*.

Extragerea şi executarea instrucțiunilor are loc sub controlul **dispozitivului central de comandă**. Pentru aceasta, prin magistrala de adrese se indică adresa locației memoriei interne, iar prin magistrala de comenzi se transmit semnalele necesare de scriere sau citire. Datele citite sau scrise se transmit prin magistrala de date.

Microprocesoarele se apreciază după următorii parametri:
– lungimea cuvântului;
– frecvența ceasului de sistem;
– capacitatea magistralelor.

Lungimea cuvântului reprezintă numărul de biți ai succesiunilor binare care pot fi memorizate în registre şi prelucrate de dispozitivul aritmetic al microprocesorului. Microprocesoarele moderne au lungimea cuvântului de 32, 64 şi mai mulți biți.

Frecvența ceasului de sistem reprezintă numărul de impulsuri pe secundă pe care le produce generatorul de impulsuri de tact din componența dispozitivului de comandă. Frecvența ceasului de sistem se măsoară în **Megahertz** (1 *MHz* = $= 10^6$ *Hz*). Întrucât impulsurile de tact sincronizează executarea microoperațiilor în toate dispozitivele microprocesorului, frecvența ceasului de sistem caracterizează rapiditatea microprocesorului.

Capacitatea magistralei reprezintă numărul de biți ai succesiunilor binare transmise prin magistrală. De obicei, capacitatea magistralei de date este egală sau mai mare decât lungimea cuvântului microprocesorului. În caz contrar, pentru a transmite un cuvânt, sunt necesare mai multe cicluri ale magistralei de date.

Capacitatea magistralei de adrese determină spațiul de adrese care pot fi accesate direct de microprocesor. Astfel, un microprocesor, având capacitatea magistralei de adrese de 16 biți, poate accesa 2^{16} locații ale memoriei interne, iar un microprocesor cu capacitatea magistralei de adrese de 32 de biți poate accesa direct 2^{32} locații.

În ansamblu, lungimea cuvântului, frecvența ceasului de sistem şi capacitatea magistralelor determină **capacitatea de prelucrare a microprocesorului**, care se măsoară în ***Mips*** – Megainstrucțiuni pe secundă. Pentru exemplificare, în *tabelul 6.2* sunt prezentate caracteristicile principale ale microprocesoarelor din familia *Intel*.

Tabelul 6.2

Caracteristicile principale ale microprocesoarelor din familia *Intel*

Microprocesor	Lungime cuvânt, *bit*	Frecvență, *MHz*	Capacitate de prelucrare, *Mips*
4040 (anul 1974)	4	0,7	0,06
8085 (anul 1976)	8	5	1,25
80286 (anul 1982)	16	16	8
Pentium 4 (anul 2004)	32	3800	2500
Pentium D (anul 2005)	64	3400	4000
Intel Core i9 (anul 2017)	64	4500	5500

Din tabelul de mai sus se observă că pe parcursul anilor 1974–2017 lungimea cuvintelor cu care operează un microprocesor a crescut de 8 ori, frecvența ceasului de sistem – de circa 6 500 de ori, iar capacitatea de prelucrare a acestora – de circa 9 200 de ori. Capacitatea de prelucrare a unui microprocesor modern este de circa 6 000 de ori mai mare decât a calculatorului *Modulului Lunar Apollo*, cu care în anul 1969 primii oameni de pe Pământ au aselenizat pe Lună.

Tipul microprocesorului şi frecvența ceasului de sistem al calculatorului personal cu care lucrați dvs. poate fi aflat cu ajutorul meniului contextual al pictogramei „My computer". Amintim că acest meniu se afişează pe ecran cu ajutorul unui clic-dreapta pe pictograma respectivă.

Întrebări şi exerciţii

❶ Explicați destinația componentelor unui microprocesor (*fig. 6.15*).

❷ Care sunt parametrii principali ai unui microprocesor? Explicați semnificația fiecărui parametru.

❸ EXPERIMENTEAZĂ! Determinați tipul şi parametrii principali ai microprocesorului din componența calculatorului personal cu care lucrați dvs.

❹ EXPLOREAZĂ! În afară de Intel, în calculatoarele moderne se utilizează şi microprocesoare produse de alte companii. Analizând informațiile despre calculatoarele personale comercializate în țara noastră, identificați producătorii de microprocesoare din componența acestora. Observați cum variază costul calculatoarelor personale în funcție de microprocesoarele utilizate.

❺ CREEAZĂ! Elaborați un mic studiu despre microprocesoarele din componența calculatoarelor de tip *desktop*, *notebook/laptop*, tabletă şi a celor încorporate în telefoanele inteligente, frecvent utilizate în țara noastră.

❻ CERCETEAZĂ! Formulată în anul 1965, „legea" lui *Moore*, fondatorul companiei Intel, descrie o tendință pe termen lung în istoria calculatoarelor: numărul de tranzistori care pot fi plasați pe un circuit integrat se dublează aproximativ la fiecare doi ani. Această tendință a continuat de mai bine de o jumătate de secol. Analizând datele referitoare la numărul de tranzistori din componența circuitelor integrate, publicate pe Internet, aflați dacă această „lege" descrie corect evoluția microprocesoarelor din ultimii ani.

Capitolul 7

REȚELE DE CALCULATOARE

7.1. Introducere în rețele

Odată cu extinderea domeniilor de aplicare a calculatoarelor, a crescut şi numărul utilizatorilor ce doreau să aibă acces la mijloace eficiente de prelucrare şi stocare a unor informații comune.

De exemplu, în cazul proiectării unei clădiri, mai mulți specialişti – arhitectul, inginerul, pompierul ș.a., doresc să aibă acces şi să introducă, dacă e necesar, modificări în desenele tehnice ale construcției în curs de elaborare. În cazul unei companii de transporturi aeriene, biletele la una şi aceeaşi cursă pot fi vândute de mai multe agenții aflate în orașe diferite.

Soluția inițială la astfel de probleme a constituit-o conectarea la un calculator central, de mare capacitate, a mai multe terminale (*fig. 7.1*).

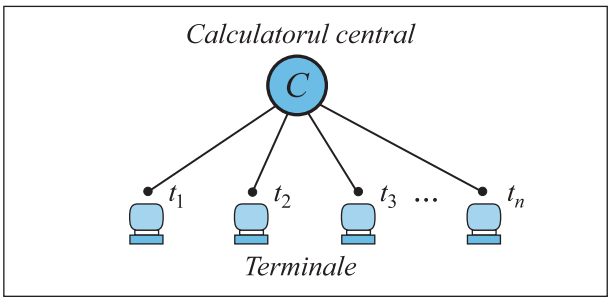


Fig. 7.1. Sistem centralizat de calcul

De regulă, un terminal constă dintr-un vizualizator, o tastatură şi, dacă e necesar, o imprimantă. Neajunsul principal al unui sistem centralizat de prelucrare a datelor este fiabilitatea redusă şi utilizarea ineficientă a resurselor de calcul.

Cu timpul, a apărut tendința de trecere de la sistemele centralizate la instalarea de calculatoare la fiecare utilizator şi asigurarea unor legături de comunicație eficientă între ele (*fig. 7.2*).

Numim rețea de calculatoare o mulțime de calculatoare ce pot schimba informații prin intermediul unei structuri de comunicație.

Calculatoarele unei rețele se conectează la structura de comunicație prin intermediul unor unități de intrare-ieşire dedicate, numite **adaptoare de rețea**. Evident, în cadrul unei rețele fiecare calculator, mai exact, fiecare adaptor de rețea are o adresă unică, denumită **adresă de rețea**.

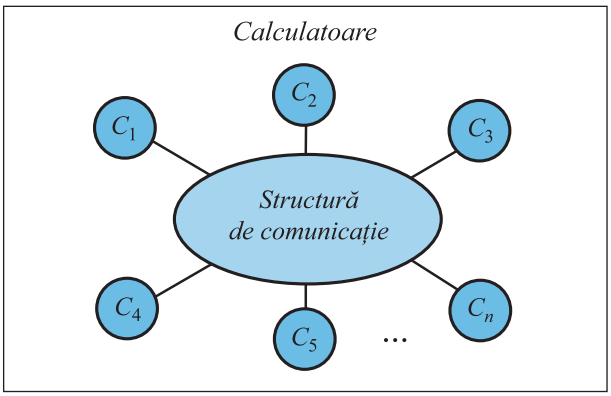


Fig. 7.2. Rețea de calculatoare

De exemplu, o rețea de calculatoare poate fi construită utilizând ca **structură de comunicație** rețeaua existentă de telefoane. În acest caz, adaptorul de rețea va include un modulator pentru conversiunea semnalelor digitale furnizate de calculator în semnale telefonice şi un demodulator pentru operația inversă. Dispozitivul respectiv poartă denumirea de **modem** (**mo**dulator-**dem**odulator). Adresa de rețea este dată de numărul de telefon al postului la care este conectat modemul.

În general, o structură de comunicație este formată din **linii de transmisie** a semnalelor. Aceste linii pot fi:

– cabluri cu fire torsadate;
– cabluri coaxiale;
– cabluri optice;
– linii cu microunde (terestre sau prin satelit).

Cablurile cu fire torsadate sunt asemănătoare celor telefonice şi asigură o capacitate de transmisie de până la 1 *Mbit*/s. **Cablurile coaxiale**, asemănătoare celor din rețelele de televiziune prin cablu, asigură o capacitate de transmisie de până la 1 *Gbit*/s. **Cablul optic** constă din fibre de sticlă sau din plastic transparent, acoperite cu un înveliş de protecție. Semnalul optic, emis de o sursă laser, se propagă prin fibră şi este recepționat de o celulă fotosensibilă. Capacitatea de transmisie a unui cablu optic poate ajunge la valoarea de 1 *Tbit*/s.

Liniile cu microunde sunt formate din stații de retransmisie ce operează în banda de unde centimetrice. Pe Pământ aceste stații se amplasează în raza vizibilității directe a antenelor, la o distanță de 40–50 de kilometri una de alta. În cazul liniilor cosmice, stațiile respective se amplasează pe sateliți. Capacitatea de transmisie a liniilor cu microunde este de ordinul 10 *Gbiți*/s.

În funcție de **aria de răspândire** a calculatoarelor dintr-o rețea, există următoarele **tipuri de rețele**:

– rețele locale;
– rețele regionale;
– rețele globale.

În **rețelele locale** calculatoarele au o arie mică de răspândire (până la 2 km) şi deservesc o singură instituție. Rețelele locale sunt formate, de regulă, din calculatoarele care se află în aceeaşi clădire sau într-un grup de clădiri. De obicei,

ca linii de transmisie se utilizează cablurile cu fire torsadate şi cablurile coaxiale.

În ultimii ani, în cazul rețelelor locale mici şi foarte mici, o răspândire tot mai largă o au mijloacele ce asigură comunicarea echipamentelor digitale prin tehnologii radio (*Wi-Fi*, *Bluetooth*) sau raze infraroşii.

Rețelele regionale acoperă aria unui oraş sau a unui sector. Liniile de comunicație se realizează prin cabluri coaxiale sau stații mici de transmisie/recepție, denumite **radiomodemuri**.

Rețelele globale acoperă suprafața unei țări, suprafața unui continent sau chiar suprafața mai multor continente. Ca linii de transmisie se utilizează cablurile optice şi liniile cu microunde (terestre sau prin satelit).

Avantajul principal al rețelelor constă în **partajarea** sau, cu alte cuvine, **utilizarea în comun** a datelor, a programelor şi a calculatoarelor din rețea.

De exemplu, în cazul unei rețele locale, pot fi partajate fişierele, discurile de capacitate mare, imprimantele, cititoarele de desene şi alte periferice. Evident, fiind accesibile pentru mai mulți utilizatori, echipamentele periferice respective vor fi utilizate mai eficient. Totodată, specialiştii instituției în cauză pot lucra în echipă asupra unor proiecte comune: bugetul anual, planul de vânzări, actualizarea bazelor de date etc.

În cazul rețelelor globale, colective de cercetători din diferite țări pot efectua calcule complexe pe un supercalculator unic în lume sau analiza în comun rezultatele unui experiment ştiințific foarte costisitor. Pe baza rețelelor examinate sunt create diverse servicii: transferul de fişiere, poşta electronică, difuzarea noutăților, conversații pe grupuri de interese, jocuri electronice, publicitate, transferul banilor etc.

Întrebări şi exerciții

❶ Numiți factorii care au contribuit la apariția rețelelor de calculatoare.

❷ Care sunt neajunsurile sistemelor centralizate de calcul?

❸ Numiți componentele principale ale unei rețele de calculatoare.

❹ Explicați destinația structurii de comunicație.

❺ Care sunt funcțiile adaptorului de rețea? Cum se identifică calculatoarele din componența unei rețele? Determinați tipul adaptorului de rețea cu care lucrați dvs.

❻ Din ce este formată o structură de comunicație?

❼ Care este destinația unui modem? Dar a unui radiomodem?

❽ Numiți capacitățile de transmisie a următoarelor linii de comunicație:
– cablu cu fire torsadate;
– cablu coaxial;
– cablu optic;
– linie cu microunde.

❾ Estimați durata de transmisie a unui film video (≈800 *Gbiți*) prin liniile de comunicație pe care le cunoaşteți dvs.

❿ EXPERIMENTEAZĂ! Determinați tipul liniilor de comunicație din structura rețelelor cu care lucrați dvs.

⓫ Cum se clasifică rețelele în funcție de aria de răspândire?

⓬ EXPERIMENTEAZĂ! Determinați tipul rețelei (locală, regională sau globală) cu care lucrați dvs.

⓭ Care sunt avantajele rețelelor de calculatoare? Ce servicii oferă o rețea de calculatoare?

⓮ EXPLOREAZĂ! Aflați capacitățile de transmisie a următoarelor mijloace de comunicație din componența rețelelor de calculatoare:
– *Wi-Fi*;
– *Bluetooth*;
– cu raze infraroşii.

7.2. Tehnologii de cooperare în reţea

Resursele unei rețele de calculatoare sunt echipamentele periferice, liniile de comunicație, calculatoarele propriu-zise, fişierele, bazele de date, programele executabile etc. Utilizarea eficientă a acestor resurse presupune lucrul în comun sau, cu alte cuvinte, cooperarea calculatoarelor şi programelor ce rulează pe ele.

Numim tehnologie de cooperare modul cum este organizată funcționarea în comun a calculatoarelor şi programelor din rețea.

Cel mai frecvent în rețelele de calculatoare se utilizează tehnologiile client-server şi egal-la-egal.

În **tehnologia client-server** o resursă comună, de exemplu, imprimanta color sau discul de mare capacitate, este gestionată de un calculator dedicat, denumit **server**. Calculatorul care doreşte să aibă acces la aceste resurse se numeşte **client**. Pentru a utiliza resursele respective, orice client trimite serverului o cerere. Serverul analizează cererile primite şi, în funcție de statutul fiecărui client, le acceptă sau le respinge (*fig. 7.3*).

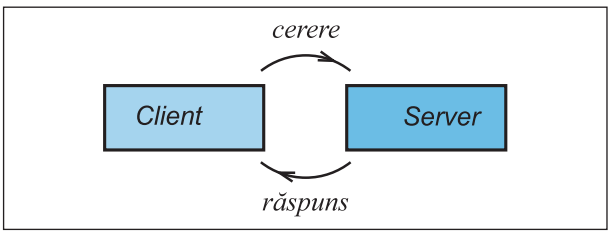


Fig. 7.3. Modelul client-server

Evident, calculatorul care gestionează fişierele comune se va numi **server de fişiere**, iar cel care gestionează imprimantele – **server de imprimare**.

Calculatorul care gestionează liniile de comunicație, alte resurse comune şi, posibil, asigură accesul la rețele externe se numeşte **server de rețea**. De regulă, anume pe acest calculator rulează şi sistemul de operare al rețelei. Celelalte calculatoare din rețea au caracteristici mai modeste şi se numesc **stații de lucru**.

Avantajele principale ale tehnologiei client-server sunt:
– utilizarea eficientă a echipamentelor scumpe;

– distribuirea rațională a lucrărilor între calculatoare în funcție de puterea lor;
– protecția sporită a datelor importante, ele fiind păstrate numai pe servere.

Cu regret, tehnologiile client-server sunt complicate şi necesită calculatoare performante.

În **tehnologia egal-la-egal** funcțiile tuturor calculatoarelor din rețea sunt identice. Fiecare calculator din rețea funcționează atât ca server, cât şi ca stație de lucru, oferind pentru uzul public resursele de care dispune, în mod curent, unele fişiere de pe discul fix, unitatea de disc optic, imprimanta etc. Fiind simplă, tehnologia dată se foloseşte în rețelele locale mici. În cazul rețelelor mari, tehnologia egal-la-egal nu permite protecția sigură a datelor.

În mod similar, tehnologiile client-server se aplică şi pentru organizarea lucrului în comun a două sau a mai multe programe.

> ***Programul în curs de execuție care oferă servicii se numeşte program server, iar programele care apelează la aceste servicii se numesc programe client.***

De exemplu, un program server ce gestionează o bază de date execută următoarele funcții:

– asigură protecția şi securitatea datelor;
– recepționează şi, dacă clientul are autorizațiile respective, execută cererile de modificare a datelor;
– recepționează cererile de citire a datelor şi, în funcție de statutul clientului, permite sau interzice accesul la datele respective;
– completează un registru în care înscrie toate operațiile efectuate în baza de date.

Programul client asigură interacțiunea utilizatorului cu baza de date şi realizează următoarele funcții:

– oferă utilizatorului o interfață simplă şi comodă;
– verifică şi editează datele introduse de utilizator;
– adresează cereri programului server;
– afişează informațiile primite din baza de date.

Programele server şi programele client pot rula pe un singur calculator sau pe calculatoare diferite. În ultimul caz, prelucrarea de date este **distribuită**. Transferul de date între programul client şi programul server se realizează prin structura de comunicație (*fig. 7.2*). Calculatorul pe care rulează un program server se numeşte **gazdă** (în engleză *host*).

De obicei, în rețelele locale programul client se execută pe stațiile de lucru, iar programele server rulează pe serverul de rețea. În cazul rețelelor regionale sau globale, pe fiecare calculator performant rulează mai multe programe server ce oferă diverse servicii programelor client de pe alte calculatoare.

Întrebări şi exerciții

❶ Explicați termenul *tehnologii de cooperare în rețea*. Ce tehnologii de cooperare în rețea cunoaşteți?

❷ Cum este organizată funcționarea calculatoarelor din rețea în cazul tehnologiei client-server?

❸ Care sunt avantajele şi dezavantajele tehnologiei client-server?

❹ Explicați destinația serverelor de rețea, a serverului de fişiere, a serverului de imprimare şi a stației de lucru.

❺ Cum este organizată funcționarea calculatoarelor din rețea în cazul tehnologiei egal-la-egal? Care sunt avantajele şi dezavantajele acestei tehnologii?

❻ EXPERIMENTEAZĂ! Determinați tehnologiile de cooperare realizate în rețeaua cu care lucrați dvs. Există oare în rețea un server de fişiere şi/sau un server de imprimare?

❼ Există în rețeaua cu care lucrați dvs. un server de rețea? Argumentați răspunsul.

❽ Este oare realizată în rețeaua cu care lucrați dvs. tehnologia egal-la-egal? Argumentați răspunsul.

❾ Cum interacționează programele în curs de execuție în cazul tehnologiei client-server?

❿ Numiți funcțiile unui program server şi ale unui program client.

⓫ Explicați termenul *calculator-gazdă*.

⓬ CERCETEAZĂ! O companie de transporturi aeriene a deschis agenții de vânzare a biletelor în mai multe oraşe din țară. Datele despre toate rutele aeriene şi locurile disponibile se păstrează în calculatorul din sediul central al companiei. Care tehnologii de rețea ar asigura lucrul eficient al agențiilor de vânzare a biletelor? Sunt oare necesare programe server şi programe client? Ce funcții ar realiza aceste programe?

⓭ ANALIZEAZĂ! Jocurile electronice colective presupun existența a 5-10 calculatoare reunite în rețea. În astfel de jocuri fiecare luptă contra tuturor. Calculatoarele respective oferă pentru uzul public o partiție de pe discul fix.
Ce tehnologie de rețea ar asigura lucrul în comun al calculatoarelor?

⓮ PROIECTEAZĂ! Elaborați o tehnologie de cooperare în rețea pentru calculatoarele din:
a) depozitele unei firme;
b) sălile de expoziție ale unui muzeu;
c) sălile de lectură ale unei biblioteci;
d) casele unui magazin care acceptă carduri bancare;
e) sistemul de eliberare a banilor lichizi prin intermediul automatelor bancare;
f) sistemul de verificare operativă a numerelor de înmatriculare a automobilelor (fiecare echipaj de poliție este dotat cu un microcalculator şi un radiomodem);
g) laboratorul de informatică.

7.3. Topologia şi arhitectura rețelelor

Structura de comunicație a unei rețele (*fig. 7.2*) asigură transferul de date între calculatoare. De obicei, datele de transmis sunt grupate în pachete.

Un **pachet de date** conține următoarele informații:
– adresa destinatarului;
– datele propriu-zise;
– informații de control;
– adresa expeditorului.

Se observă că pachetul de date poate fi tratat ca un plic obişnuit care circulă într-un sistem tradițional de poştă. Drumul parcurs de un pachet de date depinde de topologia rețelei.

Numim topologie a rețelei configurația geometrică a legăturilor dintre calculatoare.

Topologiile concrete ale rețelelor actuale sunt formate prin utilizarea structurilor de bază: stea, inel, magistrală, distribuită etc. (*fig. 7.4*).

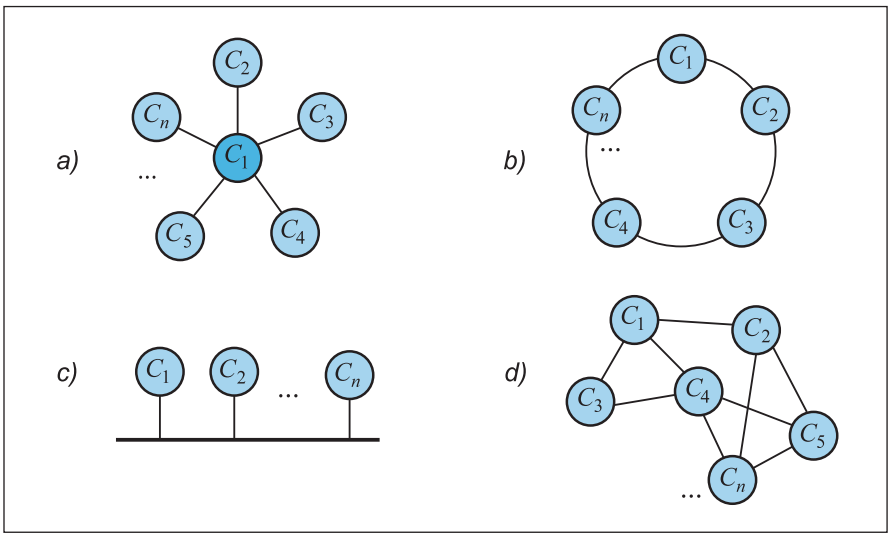


Fig. 7.4. Topologii de rețea:
a – stea; *b* – inel; *c* – magistrală; *d* – distribuită

În cazul **topologiei stea**, legătura dintre două calculatoare C_i, C_j ale rețelei are loc prin intermediul calculatorului central C_1. Din acest motiv, calculatorul C_1, denumit **calculator principal**, are un rol important în funcționarea rețelei, efectuând dispecerizarea pachetelor de date. Evident, defectarea calculatorului principal întrerupe funcționarea întregii rețele. Prin urmare, calculatorul principal trebuie să fie foarte fiabil.

În **topologia inel** legăturile dintre calculatoare formează o buclă închisă. Pachetul expediat de calculatorul C_i este transmis calculatorului C_{i+1}, care, la rândul său, îl transmite calculatorului C_{i+2} etc., până când pachetul ajunge la calculatorul destinatar C_j. Întrucât defectarea oricărui calculator întrerupe funcționarea întregii rețele, toate calculatoarele C_1, C_2, ..., C_n trebuie să fie foarte fiabile.

În **topologia magistrală** există un singur canal de comunicație la care sunt conectate toate calculatoarele. Fiecare calculator „spionează” magistrala şi interceptează pachetele adresate lui. Orice calculator C_i poate expedia un pachet numai atunci când magistrala este liberă.

Rețelele bazate pe topologia de tip magistrală sunt foarte fiabile, întrucât comunicarea dintre calculatoarele C_i, C_j va avea loc chiar şi în cazul în care toate celelalte calculatoare nu funcționează.

În **topologiile distribuite** între fiecare pereche de calculatoare există mai multe căi de transmisie a datelor. De exemplu, un pachet de date expediat de calculatorul C_1 calculatorului C_n (*fig. 7.4d*) poate ajunge la destinație pe drumul $C_1 - C_2 - C_5 - C_n$, iar altul pe drumul $C_1 - C_4 - C_n$. Evident, rețeaua rămâne funcțională chiar dacă unul sau mai multe calculatoare şi linii de comunicație nu funcționează.

De regulă, topologiile de tip stea, inel sau magistrală se utilizează în cazul rețelelor locale. Rețelele regionale şi cele globale au o topologie distribuită. Integrarea rețelelor locale în rețele regionale şi rețele globale se face conform topologiilor de bază din *figura 7.4,* fiecare nod C_1, C_2, ..., C_n reprezentând o subrețea. Prin urmare, ajungem la o structură ierarhică cu o diversitate de legături.

Setul de reguli pentru gestionarea schimbului de date într-o rețea este numit protocol de comunicație sau, pur şi simplu, protocol.

Un **protocol** definește modul de adresare a calculatoarelor, lungimea şi componența pachetelor de date, algoritmul de depistare şi corectare a erorilor, modul de conectare fizică a adaptoarelor şi cablurilor de rețea etc. La apariția primelor rețele de calculatoare fiecare producător de echipamente de calcul avea propriile sale protocoale de comunicație, ceea ce făcea imposibilă interconectarea calculatoarelor de proveniențe diferite. Acest neajuns a fost înlăturat prin standardizarea protocoalelor. Amintim că **standardul** reprezintă un document în care se reglementează calitatea, caracteristicile, forma etc. ale unui produs. Standardele internaționale sunt elaborate de Organizația Internațională pentru Standardizare (*ISO – International Standards Organisation*). Un alt organism internațional care joacă un rol important în standardizarea în domeniile electronicii şi tehnicii de calcul este Institutul Inginerilor Electricieni şi Electroniști (*IEEE – Institute of Electrical and Electronics Engineers*).

Numim arhitectură a rețelei modul în care este concepută rețeaua: topologia, protocoalele de comunicație, tehnologiile de cooperare în rețea.

În continuare prezentăm cele mai răspândite arhitecturi de rețele.

Ethernet (rețea în eter) – rețele locale realizate în conformitate cu standardul *IEEE* 802.3. Utilizează o magistrală din cablu cu fire torsadate, cablu coaxial sau cablu cu fibre optice. Viteza de transmisie a datelor ajunge la 100 *Mbiți*/s. Arhitectura a fost elaborată de firmele *XEROX*, *Intel* şi *DEC*.

Token–Ring (inel cu jeton) – rețele locale realizate în conformitate cu standardul *IEEE* 802.5. Utilizează un inel din cablu cu fire torsadate sau cablu coaxial. Viteza de transmisie a datelor este de 16 *Mbiți*/s. Arhitectura a fost elaborată de firma *IBM*.

DATAKIT – rețele locale, regionale sau globale elaborate de firma *Bell Laboratories*. Sub aspect topologic, constă dintr-o mulțime de stele interconectate. În cazul cablului cu fibre optice, se atinge viteza de transmisie de 1,5 *Gbiți*/s.

SNA (*Sistem Network Arhitecture*) – o arhitectură elaborată de firma *IBM* pentru rețelele locale, regionale şi globale. Protocoalele acestei arhitecturi au stat la baza standardelor *ISO*. Topologia inițială era de tip stea, în prezent este o topologie distribuită, suportând şi rețele locale.

ARPANET – o arhitectură concepută de mai multe universități şi corporații sub egida Ministerului Apărării al SUA (*Advanced Research Projects Agency*). Arhitectura dată se bazează pe o topologie distribuită şi foloseşte diferite linii de comunicație, de la liniile telefonice până la liniile cu microunde prin satelit. Liniile respective conectează supercalculatoare separate şi diverse rețele locale sau regionale, răspândite pe aproape jumătate din suprafața terestră. Arhitectura *ARPANET* include următoarele protocoale:

– protocolul ***IP*** (*Internet Protocol*) destinat interconectării rețelelor locale, regionale şi globale;

– protocolul serviciilor bazate pe conexiuni, numit ***TCP*** (*Transmission Control Protocol*);

– un protocol de transfer de fişiere, numit ***FTP*** (*File Transfer Protocol*);

– un protocol pentru poşta electronică, numit ***SMTP*** (*Simple Mail Transfer Protocol*);

– un protocol de ataşare de la distanță a calculatoarelor, numit ***TELNET***.

Pe baza arhitecturii *ARPANET* a fost concepută rețeaua globală de calculatoare *Internet*, care va fi studiată în paragraful următor.

Accentuăm faptul că odată cu încorporarea microcalculatoarelor în cele mai diverse maşini, aparate şi dispozitive, în particular, în cele electrocasnice, a apărut posibilitatea conectării acestora în rețele, denumite *rețele de obiecte* sau *Internet al obiectelor* (în engleză *Internet of Things*, abreviat *IoT*).

Obiectele conectate la *Internetul obiectelor* pot fi controlate la distanță, fapt ce oferă noi posibilități de creştere a productivității muncii şi îmbunătățirii vieții oamenilor. Totodată, răspândirea tot mai largă a acestor tehnologii este însoțită de apariția unor probleme importante, legate de securitatea rețelelor şi obiectelor conectate la ele. Eventualele amenințări de securitate se referă nu doar la confidențialitatea şi integritatea informațiilor procesate, dar şi la eventualele daune materiale şi umane, care pot fi provocate de obiectele ieşite de sub control sau de cele controlate de intruşi.

Întrebări şi exerciții

❶ Explicați termenul *pachet de date*. Ce informații conține un pachet de date?

❷ Explicați termenul *topologia rețelei*.

❸ STUDIU DE CAZ. Care sunt topologiile de bază ale rețelelor? Numiți avantajele şi dezavantajele fiecărei topologii de bază.

❹ Explicați cum se transmit pachetele de date în următoarele topologii de bază: stea, inel, magistrală, distribuită.

❺ Precizați drumul pe care îl va parcurge un pachet trimis de calculatorul C_2 calculatorului C_4 (*fig. 7.4a*).

❻ Găsiți drumul pe care îl parcurg pachetele expediate de calculatorul C_2 calculatorului C_4 (*fig. 7.4b*).

❼ Câte căi de transmitere a pachetelor există între calculatoarele C_1, C_n din *figura 7.4d*? Care este calea cea mai scurtă?

❽ Ce se va întâmpla cu rețelele din *figura 7.4* dacă iese din funcție calculatorul C_1? Dar dacă iese din funcție calculatorul C_2?

❾ EXPERIMENTEAZĂ! Determinați topologia rețelei locale cu care lucrați dvs. Enumerați avantajele şi dezavantajele acestei topologii.

❿ Care este destinația unui protocol? Ce norme conține un protocol?

⓫ Argumentați necesitatea standardizării protocoalelor. Cine elaborează standardele respective?

⓬ EXPERIMENTEAZĂ! Aflați protocoalele utilizate de rețeaua locală cu care lucrați dvs.

⓭ Explicați modul cum se integrează rețelele locale în rețelele regionale şi cele globale. Desenați topologia rețelelor regionale sau globale la care aveți acces dvs.

⓮ Explicați termenul *arhitectura rețelei*.

⓯ Dați exemple de arhitecturi ale rețelelor locale, regionale şi globale.

⓰ Caracterizați arhitectura rețelei cu care lucrați dvs.

⓱ Enumerați protocoalele utilizate în arhitectura *ARPANET*. Sunt oare aplicate aceste protocoale în rețeaua cu care lucrați dvs.?

⓲ Explicați termenul *Internetul obiectelor*.

⓳ EXPLOREAZĂ! În baza informațiilor publicate în Internet, aflați cum şi în ce măsură *Internetul obiectelor* influențează procesele tehnologice şi viața cotidiană a oamenilor.

⓴ CREEAZĂ! Elaborați un mic eseu care ar pune în evidență oportunitățile oferite de *Internetul obiectelor* şi modalitățile de depăşire a obstacolelor cu care el se confruntă.

7.4. Reţeaua *Internet*

Rețeaua globală *Internet* se bazează pe o topologie distribuită şi include calculatoare separate, subrețele locale, regionale sau globale (*fig. 7.5*).

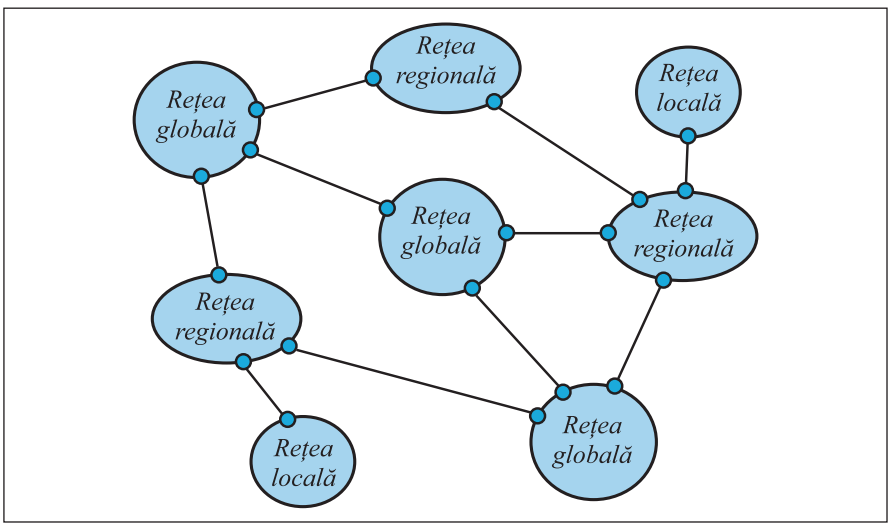


Fig. 7.5. Topologia *Internetului*

Interconectarea rețelelor se realizează cu ajutorul unor echipamente dedicate de rețea, denumite porți sau rutere. **Poarta** (*gateway*) este un calculator specializat destinat interconectării a două rețele ce utilizează protocoale total diferite de comunicație. **Ruterul** (*router*) este un calculator dedicat care interconectează rețelele ce folosesc protocoale identice şi sunt utilizate pentru a determina cea mai bună cale de transmitere a pachetelor. Calculatoarele conectate la *Internet* se numesc **gazde** (*host*). Funcționarea rețelei este reglementată de circa 100 de protocoale.

Identificarea calculatoarelor în cadrul rețelei se face cu ajutorul **adreselor *Internet***. Acestea pot fi de două tipuri: adrese numerice şi adrese simbolice.

O **adresă numerică** este formată din 32 de cifre binare (4 octeți) şi are structura prezentată în *figura 7.6*.

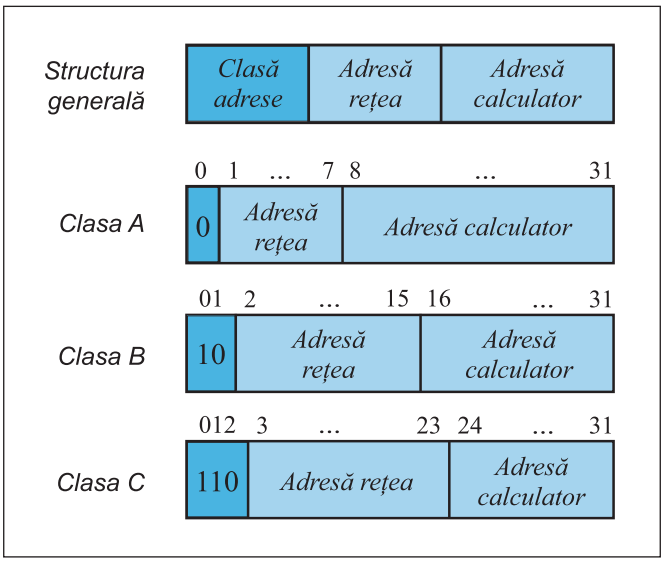


Fig. 7.6. Structura adreselor numerice

Întrucât *Internetul* este o „rețea de rețele", adresa numerică conține adresa subrețelei (câmpul *Adresă Rețea*) şi adresa calculatorului în cadrul subrețelei (câmpul *Adresă Calculator*). În funcție de numărul maxim de calculatoare pe care le poate identifica în cadrul unei subrețele, adresele se împart în clasele *A*, *B* şi *C*. Adresele din clasele *D* şi *E* au o destinație specială. Caracteristica adreselor *Internet* este prezentată în *tabelul 7.1*.

Tabelul 7.1

Caracteristica adreselor numerice

Clasa	Numărul de adrese disponibile	
	de rețea	de calculatoare
A	2^7=128	2^{24} = 16 777 216
B	2^{14}=16 384	2^{16} = 65 536
C	2^{21}=2 097 152	2^8 = 256

Adresele din clasa *A* sunt distribuite rețelelor mari, în special rețelelor globale. O astfel de rețea poate include circa 16 milioane de calculatoare. Adresele din clasa *B* sunt distribuite rețelelor medii, în special rețelelor regionale. O astfel de rețea poate include circa 65 mii de calculatoare. Adresele din clasa *C* sunt rezervate rețelelor relativ mici, care includ până la 256 de calculatoare. Fiecare adresă *Internet* este unică. Adresele sunt atribuite calculatoarelor de **Centrul Informațional al Rețelei** (*Network Information Center*).

De exemplu, numărul binar

10010010 00110011 00001001 11110111

este o adresă de clasa *B* şi identifică calculatorul numărul

00001001 11110111

din cadrul rețelei

010010 00110011.

Pentru comoditate, adresele binare se exprimă în formă zecimală, transformând separat fiecare octet. Numerele zecimale corespunzătoare fiecărui octet se delimitează prin puncte.

În cazul exemplului de mai sus obținem:

$(10010010)_2=(146)_{10}$;

$(00110011)_2=(51)_{10}$;

$(00001001)_2=(9)_{10}$;

$(11110111)_2=(247)_{10}$.

Prin urmare, în formă zecimală adresa respectivă va fi 146.51.9.247.

Adresele în formă zecimală şi cu atât mai mult cele în formă binară sunt incomode pentru publicul larg. Din acest motiv, mai frecvent sunt utilizate adresele simbolice.

O **adresă simbolică** este formată din numele calculatorului-gazdă şi nume de domenii separate prin puncte. **Domeniul** reprezintă un grup de calculatoare organizate tematic sau geografic. Orice domeniu poate fi împărțit în subdomenii, ajungându-se astfel la o structură ierarhică. Numele de domenii se indică în ordinea creşterii ariei de cuprindere.

De exemplu, adresele simbolice

c1.lme.ch.md

c5.lme.ch.md

specifică calculatoarele c1 şi c5 din domeniul lme (Liceul „Mihai Eminescu”).

Adresele simbolice

c1.lic.ch.md

c9.lic.ch.md

specifică calculatoarele c1 şi c9 din domeniul lic (Liceul „Ion Creangă”). Domeniile lme şi lic sunt subdomenii ale domeniului ch (Chişinău). La rândul său, ch este un subdomeniu al domeniului md (Republica Moldova).

În mod similar, adresele simbolice

```
rector.ase.men.ro
```

```
decan.ase.men.ro
```

specifică calculatoarele rector şi decan din domeniul ase (Academia de Studii Economice). Domeniul ase este un subdomeniu al domeniului men (Ministerul Educației Naționale), iar men este un subdomeniu al domeniului ro (România).

În mod normal, domeniul de cel mai înalt nivel este țara (md, ro, us etc.) sau tipul instituției (com – comercială, mil – militară, edu – de educație etc.).

Relațiile de incluziune între domenii pot fi reprezentate cu ajutorul diagramelor *Euler*, frecvent utilizate în teoria mulțimilor. Pentru exemplificare, în *figura 7.7* sunt prezentate astfel de diagrame pentru unele adrese simbolice din domeniul md.

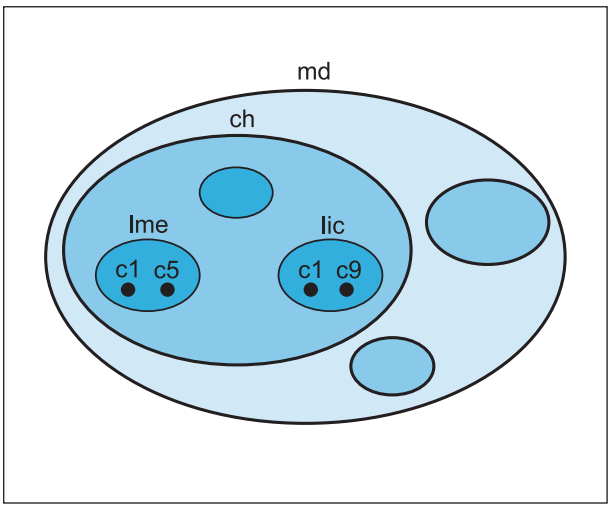


Fig. 7.7. Diagrame *Euler* pentru adrese simbolice

Pentru a **transforma** adresele simbolice în adrese numerice şi invers, în fiecare domeniu există un **server de nume** (*name server*). Acest program gestionează domeniul respectiv fără intervenția serverelor ierarhic superioare. Prin urmare, în *Internet* nu există un calculator central care ar fi unicul responsabil de cele aproximativ 10^{10} adrese de rețea.

În cazul exemplului de mai sus (*fig. 7.7*), adresele calculatoarelor c1 şi c5 sunt procesate în serverul lme, iar adresele calculatoarelor c1 şi c9 – de serverul lic. Serverul ch va lucra numai cu adresele serverelor lic şi lme, fără să se implice în procesarea adreselor de calculatoare din aceste domenii. În mod similar, serverul men nu procesează adresele calculatoarelor rector şi decan, lăsând acest lucru în obligația serverului ase.

Întrebări şi exerciţii

❶ Desenați topologia rețelei *Internet*. Cum se interconectează subrețelele în cadrul *Internetului*?

❷ Care este destinația unei porți? Dar a unui ruter?

❸ Cum se identifică calculatoarele în *Internet*? Care sunt avantajele şi dezavantajele adreselor numerice? Dar ale adreselor simbolice?

❹ Care sunt clasele de adrese numerice? Pentru ce este necesară o astfel de clasificare?

❺ Explicați destinația câmpurilor *Adresă rețea* şi *Adresă calculator* ale unei adrese numerice.

❻ Cine atribuie adrese calculatoarelor din *Internet*?

❼ Determinați clasele următoarelor adrese. Precizați adresa subrețelei şi adresa calculatorului în subrețea.

a) 45.201.19.63;
b) 201.165.213.91;
c) 154.36.79.200;
d) 192.109.58.170;
e) 15.21.207.250;
f) 217.15.69.113.

❽ Care sunt criteriile de grupare a calculatoarelor în domenii? Numiți domeniile de cel mai înalt nivel.

❾ Sunt date următoarele adrese simbolice:

a) c1.lme.ch.md;
b) c3.lme.ch.md;
c) c1.lic.ch.md;
d) director.lic.ch.md;
e) elev1.lic.ch.md;
f) c4.lme.ch.md;
g) c5.lme.ch.md;
h) c9.lic.ch.md;
i) prof.lic.ch.md;
j) elev4.lic.ch.md.

Precizați domeniile de calculatoare şi relațiile de incluziune între domenii. Desenați diagramele *Euler* pentru adresele în cauză.

❿ Desenați o diagramă *Euler* pentru adresele simbolice ce urmează. Precizați domeniile de calculatoare şi relațiile respective de incluziune.

a) rector.ase.men.ro;
b) decan.ase.men.ro;
c) student.info.ase.men.ro;
d) rector.ase.met.md;
e) decan.ase.met.md;
f) student.cib.met.md.

⓫ Care este destinația unui server de nume? Ce adrese sunt procesate de un astfel de server?

⓬ Precizați serverele de nume pentru adresele din exercițiile 9 şi 10. Indicați adresele pe care le procesează fiecare server.

⓭ EXPERIMENTEAZĂ! Aflați adresa numerică şi adresa simbolică ale calculatorului cu care lucrați dvs. Precizați clasa de adresă, adresa subrețelei şi adresa calculatorului în cadrul subrețelei. Desenați o diagramă *Euler* ce reprezintă relațiile de incluziune între domeniile la care aparține calculatorul dvs.

7.5. Servicii *Internet*

Rețeaua *Internet* oferă următoarea gamă de servicii:
– accesul calculatoarelor la distanță;
– transferul de fişiere;
– poşta electronică;
– ştiri şi discuții;
– prezentarea şi căutarea informațiilor etc.

Cooperarea calculatoarelor şi programelor care oferă aceste servicii se bazează pe modelul client-server. De obicei, pe calculatorul beneficiarului de serviciu rulează programul client, iar pe calculatorul furnizorului de servicii rulează programul server.

Serviciul Telnet permite utilizatorului să aibă acces la calculatoarele aflate la distanță. După stabilirea conexiunii, calculatorul utilizatorului devine un simplu terminal al calculatorului aflat la distanță. În continuare, utilizatorul poate lansa în execuție pe calculatorul respectiv diverse programe, poate vizualiza fişiere, schimba directoare etc. Protecția calculatoarelor şi a datelor respective se asigură prin utilizarea parolelor. Serviciul Telnet se utilizează pentru folosirea în comun a unor resurse foarte scumpe, de exemplu, a supercalculatoarelor.

Serviciul transfer de fişiere sau, mai scurt, **serviciul FTP** (*File Transfer Protocol*) permite utilizatorului să copie fişiere de pe calculatoare situate în diverse puncte geografice. Acest serviciu oferă două moduri de transfer al fişierelor:

– modul binar, în care se păstrează secvența de biți a fişierului, astfel încât originalul şi copia sunt identice bit cu bit;
– modul text, în care se transferă seturi de caractere în codul *ASCII*.

În general, pentru a avea acces la serverul FTP, clientul trebuie să introducă o parolă. Totuşi există servere publice (*FTP anonymous*) care permit accesul la fişiere fără a fi nevoie de o parolă specială.

Serviciul de poştă electronică (*electronic mail* sau, prescurtat, *e-mail*) a copiat modul de funcționare a poştei obişnuite.

Scrisoarea electronică, denumită **mesaj** (*message*) include:
– adresa destinatarului;
– subiectul, exprimat în câteva cuvinte;
– adresa expeditorului;
– textul scrisorii;
– fişiere ataşate opțional.

Fişierele ataşate pot fi de orice natură: texte, imagini, programe etc.

Scrisorile sunt depuse în fişiere speciale, denumite **cutii poştale** (*mail box*). Adresa unei cutii poştale are forma:

<nume cutie>@<Adresă calculator>,

unde:

<nume cutie> este denumirea cutiei poştale, de obicei acesta este numele de familie al utilizatorului sau o abreviere;

@ – simbolul „at” (la);

<Adresă calculator> – adresa simbolică a calculatorului client pe care este creată cutia poştală.

Exemple:

1) petrescu@c1.lme.ch.md

2) florea@director.lic.ch.md

3) ionescu@c1.lme.ch.md

4) barbu@director.lic.ch.md

Cititorii pot expedia scrisori autorilor acestui manual la adresa:

Anatol_Gremalschi@yahoo.com

Mesajele sunt transmise prin rețea de serverele de poştă care au rolul oficiilor şi centrelor poştale tradiționale.

Serviciul de poştă electronică este foarte popular datorită avantajelor sale incontestabile, şi anume: viteză, posibilitatea de a ataşa la scrisori fişiere de orice natură, facilități avansate de redactare.

Cel mai modern serviciu de prezentare şi căutare a informațiilor în *Internet* este **serviciul WWW** (*World Wide Web* – Pânza Mondială de Păianjen). În acest serviciu informația este prezentată în formă de pagini Web.

Pagina Web este un fişier scris în limbajul HTML (*Hypertext Markup Language* – Limbaj pentru marcarea hipertextului) şi poate conține, în afară de informații propriu-zise, referințe la alte pagini Web. Paginile referite se pot afla pe acelaşi calculator sau pe calculatoare situate în diverse puncte geografice (*fig. 7.8*).

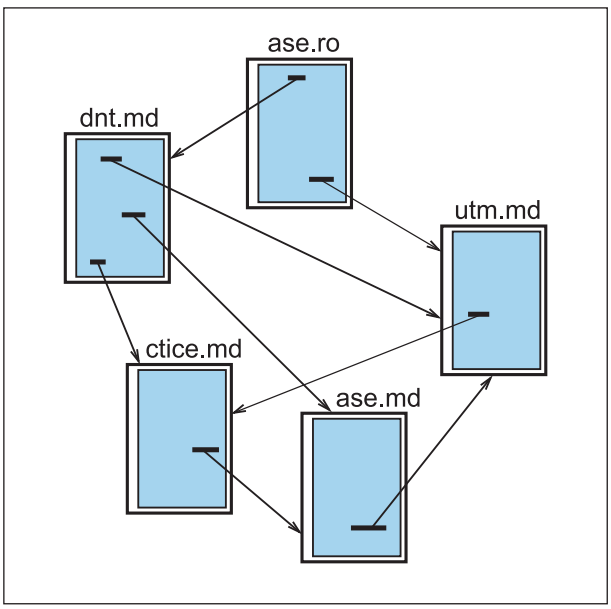


Fig. 7.8. Pagini Web

Tehnologia de cooperare în cadrul serviciului WWW este de tipul client-server. Utilizatorul care doreşte să ofere publicului larg anumite informaţii instalează pe calculatorul său un program server şi elaborează una sau mai multe pagini Web. Serverul interceptează cererile sosite de la alte calculatoare şi asigură accesul la paginile respective. Calculatorul pe care sunt instalate paginile Web şi serverul WWW se numeşte **site** (*site* – sediu, reşedinţă).

Programul client asigură transferul şi afişarea pe ecran a paginilor Web citite de pe diverse calculatoare din *Internet*. Imediat după pornire, el afişează propria pagină de referinţă (*home page* – pagina de acasă) şi aşteaptă indicaţiile utilizatorului. Când utilizatorul activează o referinţă, programul client stabileşte o conexiune cu serverul Web şi copiază de la el pagina specificată în referinţă. Pagina copiată este afişată pe ecran.

În continuare, utilizatorul activează o altă referinţă, programul client stabileşte din nou conexiunea cu un calculator din reţea, din nou se va citi o pagină Web etc. Cu alte cuvinte, utilizatorul „răsfoieşte" paginile Web de pe diverse calculatoare, indiferent de poziţia lor geografică. Din acest motiv programele client sunt numite **programe de răsfoire** sau **programe de explorare** (în engleză *browser* sau *explorer*).

În cadrul serviciului WWW resursele reţelei se specifică cu ajutorul unor adrese speciale, denumite adrese URL (*Uniform Resource Locator* – Locator Uniform de Resurse). Aceste adrese au forma:

<protocol>: // <Adresă simbolică>[:<port>]/<cale>/<fişier>

unde:

<protocol> – specifică denumirea protocolului pentru transferul datelor prin reţea;

<Adresă simbolică> – adresa calculatorului ce conţine fişierul respectiv;

<port> – portul de acces (opţional);

<cale>/<fişier> – specificarea fişierului.

Pentru exemplificare, prezentăm câteva adrese URL ce conţin informaţii interesante:

http://www.mecc.gov.md – site-ul Ministerului Educaţiei, Culturii şi Cercetării, Republica Moldova;

http://www.ctice.gov.md – site-ul Centrului de Tehnologii Informaţionale şi Comunicaţionale în Educaţie, Republica Moldova;

https://www.sciencemuseum.org.uk – site-ul Muzeului Ştiinţei din Londra, Regatul Unit al Marii Britanii şi Irlandei de Nord;

http://www.nasa.gov – site-ul agenţiei *NASA*, Statele Unite ale Americii.

Amintim că notaţia http specifică protocolul de transfer al hypertextelor (*Hypertext Transfer Protocol*).

În prezent, numărul fişierelor din reţeaua *Internet* este de ordinul miliardelor. Evident, nici nu poate fi vorba de căutarea informaţiei necesare prin citirea individuală a fiecărui fişier. Pentru a simplifica căutarea informaţiei, în cadrul reţelei *Internet* au fost create servere de căutare (*search engine*).

Serverul de căutare este un calculator puternic care explorează încontinuu reţeaua şi citeşte paginile Web sau alte informaţii prezentate publicului larg.

Acestea sunt clasificate în funcție de datele pe care le conțin, iar adresele lor sunt reținute în baza de date de pe server.

Programul client adresează serverului de căutare o cerere în care indică de ce fel de informații are nevoie. Serverul interoghează baza de date şi transmite clientului o listă de adrese la care pot fi găsite informațiile cerute.

Pentru exemplificare, amintim serverele de căutare frecvent utilizate:

http://www.yahoo.com – serverul YAHOO (*Yet Another Hierarchically Organized Oracle* – încă un oracol organizat ierarhic) al companiei *Yahoo! Inc.*;

http://www.google.com – serverul GOOGLE al corporației *Google Inc.*;

http://www.bing.com – serverul BING al corporației *Microsoft Inc.*;

http://www.yandex.ru – serverul YANDEX al companiei «Яндекс»;

http://www.infoseek.com – serverul Infoseek al firmei *Infoseek Corp*.

Accesul la aceste servere este gratuit.

Întrebări şi exerciţii

❶ Care este gama de servicii oferite de *Internet*? Cum colaborează calculatoarele din rețea în procesul prestării unui serviciu?

❷ Care este destinația serviciului *FTP*? Ce fişiere pot fi transferate prin acest serviciu?

❸ Studiați programul *FTP* instalat pe calculatorul dvs. Copiați câteva fişiere de pe serverele *FTP anonymous* sau de pe alte servere la care aveți acces.

❹ De ce depinde viteza de transfer a fişierelor? Determinați viteza de transfer a câtorva fişiere de pe serverele *FTP* aflate în Republica Moldova, România şi Statele Unite ale Americii.

❺ Cum se specifică adresele în cadrul serviciilor de poştă electronică?

❻ Explicați cum interacționează calculatoarele client şi serverele de poştă în cadrul serviciului respectiv.

❼ Ce informații conține o scrisoare electronică? Care dintre ele sunt opționale?

❽ Explicați cum se formează o adresă poştală. Aflați adresele poştale ale prietenilor dvs.

❾ Studiați programul de poştă electronică instalat pe calculatorul la care lucrați dvs. Verificați dacă acest program oferă următoarele facilități:
– utilizarea mai multor cutii poştale;
– clasificarea şi păstrarea scrisorilor în dosare;
– elaborarea scrisorilor conform unor şabloane;
– cifrarea şi descifrarea corespondenței;
– semnalarea momentelor când a sosit corespondența;
– verificarea faptului că scrisorile expediate au ajuns la destinație;
– utilizarea semnăturilor electronice.

❿ Care sunt avantajele poştei electronice? Poate înlocui acest serviciu poşta tradițională?

⓫ LUCRUL ÎN GRUP. Formați grupe de câte patru utilizatori ai poştei electronice şi determinați experimental viteza cu care se transmite corespondența.

⓬ Ce informații conține paginaWeb? Cum formează aceste pagini o „pânză de păianjen”?

⓭ Care este destinația unui server Web? Dar a unui client Web? Cum interacționează aceste programe?

⓮ Explicați modul de funcționare a unui client Web. Cum găsește acest program paginile Web amplasate pe diferite calculatoare?

⓯ Explicați modul de specificare a resurselor în *Internet* cu ajutorul adreselor URL. Care este semnificația câmpurilor acestor adrese?

⓰ EXPERIMENTEAZĂ! Determinați tipul programului de explorare a *Internetului* instalat pe calculatorul cu care lucrați dvs. Ce facilități oferă acest program? Citiți paginile Web adresele cărora sunt indicate în acest paragraf.

⓱ Care este destinația unui server de căutare? Ce servicii oferă un astfel de server?

⓲ EXPLOREAZĂ! Găsiți cu ajutorul unui server de căutare furnizorii de servicii *Internet* din Republica Moldova.

⓳ EXPLOREAZĂ! În afară de serviciile studiate în acest paragraf, rețeaua *Internet* oferă și alte servicii, cum ar fi: *Archie*, *Gopher*, *WAIS*, buletine de știri, discuții etc. Folosind un server de căutare, aflați informația respectivă despre aceste servicii.

⓴ PROIECTE! Elaborați paginile Web ale clasei și ale liceului dvs.

Capitolul 8

MODULE LA ALEGERE

Modulele din acest capitol sunt opționale. Ce înseamnă acest lucru? Aveți posibilitatea să alegeți unul dintre ele şi să-l studiați în ritm propriu, de sine stătător sau împreună cu colegii, în procesul exersării şi elaborării de proiecte, cu ajutorul mijloacelor de instruire asistată de calculator. După ce ați ales un modúl, împreună cu profesorul de informatică veți decide ce programe de aplicații veți utiliza, veți avea grijă ca ele să fie licențiate sau cu distribuție liberă, vă veți asigura că ele sunt instalate pe calculatoarele din laboratoarele de informatică şi pe cele personale, după caz.

Proiectele care se referă la subiecte ample, de exemplu, la orașe, sate, şcoli, vor fi elaborate în cadrul echipelor de elevi, fiecare membru al echipei specializându-se pe un anumit aspect, de exemplu, pe cel istoric, demografic, economic, cultural, etnografic etc.

Tot în echipe vor fi elaborate şi proiectele ce se caracterizează printr-un bogat conținut multimedia, specializarea membrilor echipelor realizându-se după genul materialelor respective, de exemplu, imagini, secvențe audio, secvențe video, designul interfețelor grafice, programarea algoritmilor de prelucrare a datelor ş.a.

8.1. Introducere în reţele

Aceste tehnici vor fi studiate pe etape, după cum urmează.

1. Digitalizarea informației audio. Amintiți-vă din fizică caracteristicile sunetului ca fenomen fizic şi parametrii ce influențează percepția de către om a undelor sonore. Din informatică amintiți-vă codificarea digitală a informației audio. Utilizând bogatele resurse ale Internetului, aflați cum se realizează compresia şi decompresia datelor audio, care sunt formatele fişierelor ce conțin date audio.

Acordați o atenție deosebită echipamentelor destinate înregistrării digitale a sunetului şi redării acestuia: parametrii tehnici, interacțiunea cu calculatoarele personale, costurile, domeniile de utilizare. În funcție de dotările laboratorului de informatică şi de echipamentele digitale personale de care dispuneți, de exemplu, reportofoane digitale, plăci de sunet, microfoane, difuzoare, amplificatoare, studiouri digitale multimedia etc., decideți referitor la modalitățile de creare, stocare şi redare a înregistrărilor sonore.

Exercițiile recomandate pentru această etapă sunt:

– explicarea principiilor de codificare şi decodificare a datelor audio;

– determinarea volumului datelor audio necomprimate cunoscând parametrii de codificare;

– determinarea formatului audio cunoscând extensiunile denumirilor de fişiere;
– identificarea şi explicarea parametrilor de bază şi a caracteristicilor principale ale echipamentelor frecvent utilizate în colectarea, înregistrarea, prelucrarea şi reproducerea secvenţelor audio.

Pentru o înţelegere mai aprofundată a proceselor fizice şi informatice legate de înregistrarea digitală şi redarea sunetelor, se recomandă efectuarea următoarelor *studii de caz*:
– Înregistrarea analogică şi înregistrarea digitală a sunetului.
– Analiza comparată a echipamentelor destinate prelucrărilor digitale audio.
– Analiza comparată a formatelor de fişiere audio.

Consolidarea şi aprofundarea cunoştinţelor în domeniul digitalizării informaţiei audio poate fi făcută elaborând următoarele *proiecte de cercetare*:
– Influenţa frecvenţei de discretizare a semnalelor sonore asupra calităţii de redare a acestora.
– Influenţa mărimii pasului de cuantificare a semnalelor sonore asupra calităţii de redare a acestora.
– Influenţa gradului de compresie a semnalelor sonore asupra calităţii de redare a acestora.

2. Prelucrarea înregistrărilor digitale audio. Studiaţi atent oferta de programe licenţiate şi cu distribuţie liberă, destinate prelucrărilor digitale audio. Decideţi ce program veţi utiliza şi instalaţi programul respectiv pe calculatorul personal cu care lucraţi dumneavoastră.

Studiaţi mediul de lucru şi facilităţile oferite de editorul digital audio cu care veţi lucra. Acordaţi o atenţie deosebită formatelor acceptate de fişiere audio, modalităţilor de stocare şi organizare a fişierelor audio, compatibilităţii editorului cu echipamentele externe de înregistrare şi redare a fişierelor audio. Convingeţi-vă că interfeţele grafice ale editorului digital audio sunt intuitive şi uşor de utilizat.

Experimentând cu cele mai diverse înregistrări audio digitale, formaţi-vă următoarele abilităţi de prelucrare a acestora:
– secvenţierea şi concatenarea fragmentelor audio;
– transformări de amplitudine (amplificare, mixare normalizare);
– modificarea tonului şi a duratei de redare;
– filtrarea semnalelor sonore;
– aplicarea efectelor.

În procesul prelucrării înregistrărilor digitale audio, utilizaţi facilităţile de analiză a sunetelor oferite de editoarele respective: analiza spectrului, modificarea volumului.

În scopul organizării raţionale a fişierelor audio pe purtătorii de informaţie şi al racordării fişierelor respective la cerinţele înaintate faţă de calitatea redării sunetelor digitale, efectuaţi conversii de formate audio.

Exerciţiile ce vor contribui la consolidarea cunoştinţelor teoretice şi la dezvoltarea abilităţilor practice de utilizare a echipamentelor digitale şi a programelor destinate prelucrărilor audio includ:
– utilizarea facilităţilor oferite de editoarele digitale audio;
– analiza comparată a înregistrărilor uneia şi aceleiaşi secvenţe sonore efectuate în formate audio diferite;

– analiza calității de redare a secvențelor audio în funcție de parametrii de înregistrare şi stocare;

– conversia formatelor fişierelor audio;

– transmiterea fişierelor audio prin diferite mijloace de comunicații digitale.

O sinergie a mijloacelor informatice şi a celor specifice artelor audiovizuale poate fi asigurată prin elaborarea următoarelor proiecte:

– Elaborarea fundalului audio pentru evenimentele importante din viața personală, a familiei, a clasei, a şcolii, a comunității.

– Înregistrarea şi mixarea informațiilor audio ce provin de la mai multe surse pe durata unor evenimente festive, sportive, artistice, recreative.

– Elaborarea pistelor sonore pentru filmele video despre diversele evenimente din viața şcolii şi a comunității.

3. Digitalizarea informației video. Pentru început va trebui să vă amintiți din fizică natura ondulară şi natura corpusculară a luminii şi să țineți cont de faptul că în informatică lumina este tratată ca fiind o undă electromagnetică. În continuare, aflați din Internet modul în care ochiul uman percepe lumina şi cum din mărimile parametrilor fizici ai undelor electromagnetice derivă culorile, luminanța şi contrastul imaginilor. Pentru o înțelegere mai profundă a proceselor de digitalizare a informațiilor video, va trebui să vă împrospătați cunoştințele despre metodele de cuantizare a imaginilor dinamice şi de calculare a cantității de informație conținute de acestea. Acordați o atenție deosebită frecvenței de discretizare în timp, frecvenței de discretizare în spațiu, pasului de cuantificare a semnalului video, puterii de rezoluție, numărului de culori de bază.

Pornind de la faptul că foarte des imaginile video conțin informații ce sunt redundante în cazul percepției acestora de către om, faceți cunoştință cu metodele de compresie şi decompresie a datelor video, cu formatele respective de fişiere. Identificați legătura dintre ratele de biți şi calitatea redării informațiilor video, înregistrate în formă digitală.

Consolidați cunoştințele teoretice prin studierea echipamentelor destinate înregistrării digitale a informațiilor video şi redării acesteia: parametrii tehnici, interacțiunea cu calculatoarele personale, costurile, domeniile de utilizare. În funcție de dotările laboratorului de informatică şi de echipamentele digitale personale de care dispuneți, de exemplu, camere video digitale, plăci video, obiective, vizualizatoare, proiectoare multimedia, studiouri digitale multimedia etc., decideți referitor la modalitățile de creare, stocare şi redare a înregistrărilor digitale video.

Exercițiile recomandate pentru această etapă sunt:

– descrierea domeniilor de activitate în care se utilizează informația video;

– descrierea şi compararea parametrilor undelor electromagnetice din perspectiva percepției lor de către ființele umane;

– descrierea şi explicarea factorilor video ce pot periclita sănătatea;

– explicarea principiilor de codificare şi decodificare a datelor video;

– determinarea volumului datelor video necomprimate cunoscând parametrii de codificare;

– determinarea formatului video cunoscând extensiunile denumirilor de fişiere;

– identificarea şi explicarea parametrilor de bază şi a caracteristicilor principale

ale echipamentelor frecvent utilizate în colectarea, înregistrarea, prelucrarea şi reproducerea secvenţelor video.

Studiile de caz, propuse pentru a fi efectuate de către elevi, sunt:

– Înregistrarea analogică şi înregistrarea digitală a informaţiilor video.

– Percepţia diverselor unde electromagnetice de către fiinţa umană.

– Impactul modificării parametrilor undelor electromagnetice asupra percepţiei acestora de către fiinţa umană.

Proiectele de cercetare listate mai jos vor contribui la înţelegerea mai profundă a modului în care realizările teoretice ale informaticii influenţează tehnologiile de colectare, înregistrare, stocare, prelucrare şi redare a informaţiilor video:

– modul în care variaţia frecvenţei de discretizare a semnalelor video influenţează calitatea de redare a acestora;

– modul în care variaţia pasului de cuantificare a semnalelor video influenţează calitatea de redare a acestora;

– influenţa gradului de compresie a semnalelor video asupra calităţii acestora.

4. Prelucrări digitale video. Pe piaţa de produse soft şi în distribuţie liberă sunt promovate cele mai diverse programe de calculator, destinate prelucrării înregistrărilor video. Utilizatorii pot alege atât din produsele destinate profesioniştilor, cât şi din cele destinate amatorilor. De obicei, sistemele de operare dispun de aplicaţii destinate redării înregistrărilor video, iar unele dintre ele – chiar şi de editoare video.

Prin urmare, înainte de a parcurge această etapă, fiecare elev, sub îndrumarea profesorului de informatică, va decide ce produse-program va utiliza, ţinând cont de faptul că ele trebuie să fie compatibile cu echipamentele digitale utilizate pentru înregistrarea şi redarea informaţiilor video. De asemenea, se va ţine cont şi de faptul că editoarele digitale video necesită resurse semnificative de calcul, atât de memorie (de ordinul Giga octeţilor), cât şi de capacităţi de prelucrare (de ordinul Giga instrucţiuni pe secundă).

Studierea facilităţilor oferite de editoarele digitale video va avea drept scop formarea şi dezvoltarea următoarelor abilităţi:

– secvenţierea şi concatenarea fragmentelor video;

– montarea fragmentelor video;

– transformări temporale;

– asocierea pistei sonore;

– modificarea duratei de redare;

– filtrarea semnalelor video;

– aplicarea efectelor;

– subtitrarea secvenţelor video;

– conversia formatelor.

Exerciţiile recomandate pentru această etapă sunt:

– utilizarea principalelor facilităţi ale unui editor digital video;

– analiza comparată a înregistrărilor uneia şi aceleiaşi secvenţe efectuate în formate video diferite;

– analiza calităţii de redare a secvenţelor video în funcţie de parametrii de înregistrare şi stocare;

– conversia formatelor fişierelor video.

Tematica *proiectele recomandate* va fi racordată la cele propuse mai sus în cazul prelucrării digitale a secvențelor audio şi va include:

– Elaborarea fundalului video pentru evenimentele importante din viața personală, a familiei, a clasei, a şcolii, a comunității.

– Înregistrarea şi mixarea informațiilor video ce provin de la mai multe surse pe durata unor evenimente festive, sportive, artistice, recreative.

– Montarea filmelor video despre evenimente din viața şcolii şi a comunității.

5. Diseminarea informațiilor multimedia. La această etapă, de sine stătător sau sub îndrumarea profesorului de informatică, studiați serviciile *online*, care vă oferă posibilitatea:

– să căutați resursele multimedia dorite;

– să diseminați propriile elaborări multimedia;

– să redactați secvențele multimedia în regim *online*.

Se recomandă să elaborați un catalog al serviciilor preferate şi să studiați în detaliu fiecare dintre facilitățile serviciilor frecvent utilizate de publicul larg.

Exercițiile propuse în cadrul acestei etape sunt:

– transmiterea fişierelor multimedia prin mijloace de comunicații digitale;

– căutarea informațiilor multimedia pe Internet;

– înregistrarea şi crearea propriilor profiluri în cadrul serviciilor online;

– publicarea propriilor elaborări multimedia;

– includerea în paginile Web, personale, a clasei, a şcolii, a asociațiilor de elevi, a legăturilor către propriile resurse multimedia publicate online.

În calitate de *studiu de caz*, vă recomandăm să faceți o analiză comparată a serviciilor online de diseminare a informațiilor multimedia, identificând avantajele şi dezavantajele acestora, domeniile de aplicare în funcție de specificul informațiilor multimedia destinate diseminării în spațiul virtual.

6. Etica digitală şi respectarea dreptului de autor. Dacă în Antichitate textele inscripționate pe plăci de piatră sau scrise pe papirus erau accesibile unui număr foarte mic de oameni, odată cu apariția tiparului, numărul oamenilor ce au început să aibă acces la cărți şi, ulterior, la ziare, a crescut exponențial. Deşi radioul şi televiziunea au simplificat şi mai mult procesele de difuzare a informațiilor, instituțiile respective, adică tipografiile, stațiile de emisie radio şi TV aparțineau şi mai aparțin fie statelor, fie marilor companii. În consecință, cetățenii de rând nu prea aveau posibilitatea să-şi disemineze de sine stătător, pentru un număr cât mai mare de oameni, cărțile, fotografiile, fonogramele, filmele create de ei. Situația s-a schimbat radical odată cu apariția Internetului şi a serviciilor *online* de difuzare a produselor multimedia, care permit fiecăruia dintre noi să supună atenției marelui public toate informațiile pe care le considerăm ca fiind importante, interesante, originale sau valoroase din punct de vedere ştiințific sau artistic.

Indiscutabil, această libertate de exprimare contribuie la consolidarea şi dezvoltarea societăților democratice, însă ea cere de la fiecare dintre noi respectarea în spațiul virtual a anumitor reguli. Ați studiat aceste reguli, denumite generic *reguli de etică digitală*, începând cu clasele de gimnaziu. Totuşi, vă recomandăm cu fermitate să vă împrospătați cunoştințele în acest domeniu, să respectați cu strictețe regulile respective şi să nu uitați că într-o lume multiculturală şi poliконfesională cuvintele, imaginile, secvențele sonore şi secvențele video pot fi

interpretate şi înțelese în mod diferit. În consecință, înainte de a posta pe Internet un produs multimedia elaborat de dumneavoastră, aveți grijă ca el să transmită mesaje pozitive, să promoveze valorile general-umane, să încurajeze participarea şi incluziunea.

Un alt aspect foarte important legat de diseminarea produselor multimedia constă în necesitatea respectării dreptului de autor şi a drepturilor conexe. Amintim că dreptul de autor se extinde asupra operelor literare, artistice şi ştiințifice exprimate în următoarele forme:

– scrisă (manuscris, text dactilografiat, partitură etc.);
– orală (interpretare publică etc.);
– imprimare audio sau video (mecanică, magnetică, digitală, optică etc.);
– de imagine (desen, schiță, pictură, plan, poză etc.);
– tridimensională (sculptură, model, machetă, construcție etc.).

Prin urmare, înainte de a include în propriile elaborări unele fragmente din operele menționate mai sus, informați-vă dacă deținătorii dreptului de autor permit acest lucru.

De asemenea, nu prezentați drept creații personale ideile, fragmentele de text, imaginile, fonogramele, videogramele preluate din opere şi produse multimedia create de alte persoane, întrucât acest fapt reprezintă un plagiat. Dacă autorii operelor şi produselor multimedia respective permit preluarea de fragmente (acest lucru poate fi aflat consultând licențele respective), indicați în mod obligatoriu sursa: denumirea operei, editura care a publicat-o, referința URL ş.a.m.d.

Exercițiile recomandate pentru această etapă sunt:

– identificarea însemnelor ce declară dreptul de autor;
– explicarea regulilor ce vizează respectarea dreptului de autor;
– identificarea tipurilor licențelor de distribuție;
– utilizarea licențelor pentru distribuție.

În calitate de *studiu de caz* se recomandă analiza respectării drepturilor de autor în propriile produse multimedia, în produsele multimedia elaborate şi publicate online de colegii de clasă şi de şcoală, de către elevii din alte instituții de învățământ. Folosiți în acest scop facilitățile oferite de motoarele de căutare a informațiilor text, informațiilor grafice, informațiilor audio şi video. De asemenea, vă vor fi utile şi programele antiplagiat, disponibile *online* pe Internet.

8.2. Programarea vizuală

Istoric, primele programe au fost scrise în limbajul cod-calculator. Ulterior, s-a trecut la limbajele de asamblare şi limbajele de programare de nivel înalt. În cazul limbajelor de programare de nivel înalt, programele reprezintă texte, care, pentru a fi „înțelese” de calculator, sunt „traduse” în coduri binare.

Pe parcursul a unei lungi perioade de timp, produsele-program au fost elaborate cu ajutorul limbajelor de programare de nivel înalt, folosind la început în acest scop simple editoare de texte, iar ulterior, medii integrate de dezvoltare a programelor. Aceste medii, pe lângă editarea textelor şi introducerea rapidă de modificări, permit lansarea în execuție şi depanarea programele aflate în curs de

elaborare. Cunoaşteţi deja astfel de medii de programare şi chiar aţi elaborat şi depanat mai multe programe în ele.

Indiferent de mediile de dezvoltare a programelor utilizate fie în scopuri didactice, fie în scopuri de producţie, în ele se lucrează cu texte. Însă, odată cu creşterea complexităţii programelor, textele respective au devenit voluminoase şi greu percepute chiar şi de autorii acestora, fapt ce a stimulat dezvoltarea aşa-numitei *programări vizuale* – mijloc de elaborare a programelor nu prin scrierea de texte, ci prin manipularea cu obiecte grafice.

Cunoaşteţi deja mediul de programare vizuală *Scratch*, special elaborat pentru a studia informatica începând cu clasele primare. În acest mediu de programare elevii implementează algoritmii de comandă cu aşa-numiţii spiriduşi prin manipularea cu obiecte grafice ce reprezintă spiriduşii propriu-zişi, comenzile pe care ei trebuie să le execute şi blocurile de control ale fluxului de execuţie: ramificări în funcţie de respectarea anumitor condiţii, organizarea ciclurilor şi subprogramelor.

Mijloace clasice de programare vizuală, elaborate în scopuri de producţie, sunt aplicaţiile de calcul tabelar, studiate în clasele gimnaziale. În aceste aplicaţii, utilizatorul implementează algoritmii de prelucrare a datelor numerice nu prin scrierea de programe într-un limbaj de nivel înalt, de exemplu, în PASCAL sau C++, ci prin operarea asupra unor obiecte grafice specifice, cum ar fi foile de calcul, celulele, rândurile, coloanele, diagramele, graficele, formularele etc.

În prezent, renumitele companii ce activează în domeniul tehnologiei informaţiei şi comunicaţiilor propun o gamă foarte largă de mijloace de programare vizuală, mijloace ce uşurează foarte mult implementarea chiar şi a celor mai complecşi algoritmi. Utilizarea tot mai largă a programării vizuale deschide noi oportunităţi nu doar pentru persoanele ce doresc să facă o carieră profesională în informatică, ci şi pentru cele pasionate de artele digitale, de instruirea asistată de calculator, de elaborarea de jocuri computerizate interactive.

Prin urmare, cunoaşterea programării vizuale va permite elevilor să ia decizii argumentate referitoare la continuarea studiilor postliceale şi la digitalizarea acelor domenii de activitate profesională în care ei au de gând să se afirme. Iar pentru a facilita procesul de studiere a programării vizuale, recomandăm elevilor să parcurgă următoarele etape:

1. Selectarea mediului de programare vizuală. Evident, orice program elaborat în mediile de programare vizuală are echivalentul său într-o formă de text, scrisă într-un limbaj de programare de nivel înalt. Mai mult ca atât, pentru a implementa eficient anumiţi algoritmi de prelucrare a datelor, uneori chiar şi în programarea vizuală, se cere şi scrierea de subprograme într-un limbaj de nivel înalt. Prin urmare, la selectarea mediului de programare vizuală se va ţine cont nu doar de uşurinţa de utilizare a acestuia, dar şi de orientarea lui spre un anumit limbaj de programare de nivel înalt.

De exemplu, elevii care au studiat limbajele din familia PASCAL vor opta pentru mediile de programare vizuală *Delphi* sau *Lazarus*, iar cei ce au studiat limbajele din familia C – pentru mediile de programare vizuală *Microsoft Visual Studio* sau *Code::Blocks*.

Elevii care doresc să studieze doar conceptele de bază ale programării vizuale, fără a apela la un limbaj profesional de programare de nivel înalt, o pot face

utilizând *MIT App Inventor*, care folosește un limbaj vizual şi interfețe grafice, similare celora din mediul de programare *Scratch*.

Recomandăm elevilor, înainte de a opta pentru un anumit mediu de programare vizuală, să se familiarizeze cu clasificarea acestora, cu parametrii lor tehnici şi ergonomici, cu cerințele pe care produsele respective le înaintează față de calculatoarele ce vor fi utilizate în procesul de învățare şi capacitatea de transmisie cerută de la conexiunile Internet. De asemenea, ei ar trebui să se conducă de scopul urmărit: să-şi aprofundeze competențele în domeniul elaborării interfețelor grafice destinate aplicațiilor de consolă şi aplicațiilor WEB sau să-şi dezvolte noi competențe în domeniile aflate la intersecția tehnologiilor informației şi a artelor: animații digitale, jocuri pe calculator, realitatea augmentată etc.

2. Studierea noțiunilor şi a principiilor de bază a programării vizuale. La această etapă va trebui să vă familiarizați cu noțiunile de obiect grafic, conexiuni şi acțiuni, cu tipurile de obiecte grafice primare şi metaobiecte, cu proprietățile obiectelor. O atenție deosebită trebuie acordată înțelegerii specificului programării orientate pe evenimente în context vizual: situații, acțiuni şi rezultate.

Principalele *exerciții* recomandate pentru această etapă sunt:
- explicarea principiilor programării vizuale;
- explicarea noțiunilor de bază ale programării vizuale;
- clasificarea obiectelor/instrucțiunilor grafice;
- controlul evenimentelor, acțiunilor, stărilor.

La o înțelegere mai profundă a specificului programării vizuale vor contribui *următoarele studii de caz*:
- Instrucțiunile grafice versus instrucțiunile scrise.
- Flexibilitatea programării vizuale.
- Depanarea programelor: vizuală şi procedurală.
- Depanarea programelor: vizuală şi orientată pe obiecte.

3. Formarea şi dezvoltarea abilităților de utilizare a mediilor de programare vizuală. Folosind în calitate de instrument de învățare sistemul de asistență al mediului vizual de programare selectat la prima etapă şi instalat, după caz, pe calculator, elevii vor identifica componentele principale ale produsului-program utilizat şi specificul interfeței grafice cu utilizatorul. Vor fi studiate în detaliu structura proiectului, tipologia fişierelor din componența proiectelor şi modalitățile de organizare a acestora.

În baza proiectelor cu caracter didactic propuse de profesorul de informatică, elevii îşi vor forma abilitățile necesare pentru introducerea şi editarea programelor, rularea şi depanarea acestora. Principalele *exerciții* recomandate în acest scop sunt:
- identificarea componentelor principale ale mediului de programare vizuală;
- explicarea structurii proiectelor din cadrul mediilor de programare vizuală;
- identificarea fişierelor proiectelor;
- conectarea instrucțiunilor grafice în programe;
- depanarea programelor vizuale;
- derularea programelor vizuale.

La formarea unei viziuni profunde asupra bogatei diversități de mijloace destinate programării vizuale va contribui elaborarea următoarelor *studii de caz*:

– Avantajele şi dezavantajele mediilor de programare vizuală.
– Analiza comparativă a mediilor de programare vizuale şi tradiționale.
– Prezentarea unor medii şi limbaje larg răspândite de programare vizuală (*Alice*, *Kodu*, *Scratch*, *ToonTalk*, *Cameleon*, *Filter Forge* ş.a.).

4. Formarea şi dezvoltarea competenței de programare vizuală. Această etapă va începe cu studierea instrucțiunilor grafice şi a criteriilor de clasificare a acestora. În continuare, elevii vor face cunoştință şi vor studia în bază de mici exemple facilitățile ce asigură:
– Realizarea mişcării: deplasarea, rotirea, controlul apariției unui anumit eveniment, controlul apariției unei anumite situații.
– Setarea aspectului: redimensionarea, afişarea mesajelor, afişarea valorilor, modificarea obiectelor, setarea vizibilității, setarea straturilor.
– Controlul sunetului: setarea, start, stop, durata, aplicarea efectelor.
– Prelucrarea evenimentelor: pornirea/oprirea programelor, acționările de taste, acționările asupra butoanelor şoricelului, acționările asupra obiectelor grafice, controlul fluxurilor de mesaje.
– Controlul fluxurilor de execuție: instrucțiunile de selecție, instrucțiunile ciclice, pauze, opriri, clonări.
– Controlul situațiilor: setarea proprietăților obiectelor, introducerea datelor, setarea proprietăților unităților periferice, eşalonarea în timp.
– Crearea şi apelarea subprogramelor.
Exercițiile recomandate pentru această etapă sunt:
– deplasarea şi controlul obiectelor grafice;
– afişarea mesajelor;
– setarea proprietăților geometrice/ de culoare/ vizibilitate/ ale obiectelor grafice;
– integrarea componentelor sonore (multimedia) în programele vizuale;
– controlul obiectelor grafice dinamice cu ajutorul tastaturii şi al şoricelului;
– crearea şi apelarea subprogramelor.
Tot la această etapă va demara şi *elaborarea de proiecte*, tematica recomandată a cărora include:
– Interacțiunea dintre două sau mai multe personaje (obiecte) vizuale.
– Redarea dialogurilor între personaje.
– Jocurile elementare (prototenis, prinderea obiectelor căzătoare, urmăriri).
– Animarea poveştilor.
Elevii sunt încurajați să concretizeze de sine stătător sau sub îndrumarea profesorului de informatică tema proiectului la care ar prefera să lucreze, să decidă asupra modului de organizare a lucrului la proiecte propuse de ei, individual sau în grup, şi să elaboreze planuri de desfăşurare a lucrărilor necesare pentru realizarea proiectelor respective.

5. Prelucrarea datelor externe. În spatele oricăror obiecte grafice se află anumite date: informații referitoare la forma şi aspectul acestora, amplasarea lor în spațiu, traiectoria de deplasare în spațiul de lucru, mesajele recepționate de la alte obiecte etc. Rezultatele prelucrărilor efectuate asupra datelor respective determină comportamentul în timp şi în spațiu a obiectelor, interacțiunile dintre ele şi mediul extern.

Pentru a studia modalităţile de implementare a algoritmilor de prelucrare a datelor, elevii îşi vor împrospăta cunoştinţele referitoare la tipurile de date simple şi la tipurile de date structurate, la operaţiile ce pot fi efectuate asupra datelor respective. O atenţie deosebită va fi acordată specificului conversiei de tipuri şi operaţiilor aritmetice, relaţionale şi logice ce pot fi efectuate asupra datelor anume în mediile vizuale de programare.

În continuare, recomandăm elevilor să studieze obiectele vizuale pentru introducerea şi afişarea datelor, evenimentele şi acţiunile destinate modificării datelor: situaţiile apărute în procesul execuţiei programului, comenzile venite de la tastatură, de la şoricel, de la alte echipamente periferice.

Exerciţiile ce vor contribui la dezvoltarea abilităţilor de prelucrare a datelor în programele vizuale sunt:

– declararea şi utilizarea variabilelor ce aparţin la tipuri de date simple şi structurate;

– utilizarea în cadrul programelor vizuale a operaţiilor aritmetice, relaţionale şi logice;

– modificarea valorilor variabilelor prin acţiuni şi reacţii la evenimente (acţionare tastelor şi a butoanelor şoricelului, sosirea mesajelor de la alte obiecte grafice sau din exterior etc.);

– afişarea rezultatelor prin asocierea acestora unui obiect grafic;

– declararea şi utilizarea variabilelor ce aparţin tipurilor de date structurate;

– căutarea informaţiilor în datele structurate;

– modificarea structurii datelor.

O înţelegere mai aprofundată a specificului prelucrării datelor o poate asigura efectuarea unui *studiu de caz* în care s-ar analiza avantajele şi dezavantajele modalităţilor de implementare a algoritmilor de calcul prin programare textuală şi prin programare vizuală. Un astfel de studiu de caz ar permite şi identificarea posibilităţilor de ataşare a programelor textuale anumitor obiecte din componenţa programelor vizuale.

Proiectele recomandate pentru această etapă sunt:

– Calcularea numerelor maxime din mulţimile de numere enunţate de personaje (obiecte).

– Calcularea sumelor mulţimilor de numere enunţate de personaje.

– Găsirea numerelor prime din mulţimea celor enunţate de personaje.

– Calcularea celui mai mare divizor comun.

– Sortarea tablourilor.

În ansamblu, integrarea cunoştinţelor şi dezvoltarea abilităţilor de programare vizuală pot fi asigurate prin elaborarea unor proiecte complexe, cum ar fi interfeţele grafice pentru aplicaţiile de consolă sau aplicaţiile Web, jocuri didactice, jocuri de agrement, programe pentru instruirea asistată de calculator, filme de animaţie (anime-uri), produse cu realitate augmentată (materiale publicitare, modele dinamice, reconstituiri în dinamică a obiectelor istorice ş.a.). Elaborarea unor astfel de proiecte este posibilă prin crearea unor echipe de elevi, în care, pe lângă o cooperare strânsă, trebuie să existe şi o delimitare explicită a rolurilor şi responsabilităţilor, o specializare în funcţie de vocaţia fiecărui elev.

8.3. Limbaje de marcare a hipertextului

Posibil, în clasa precedentă ați studiat modulul opțional *Web design*. Studiind acest modúl, ați învățat cum să elaborați documente Web atât cu ajutorul aplicațiilor de uz general, de exemplu, cele de oficiu, cât şi cu ajutorul aplicațiilor specializate, denumite editoare Web. În ambele cazuri, elevii, posibil, viitorii designeri Web, au operat cu obiectele paginilor Web aflate în curs de elaborare, adică cu texte, imagini, elemente de navigare, de introducere a informațiilor, de rulare a secvențelor audio şi video etc.

Atât aplicațiile de uz general, cât şi cele specializate, destinate elaborării documentelor Web, pe care le-ați studiat în clasele precedente, lucrează în regimul WYSIWYG (*What You See Is What You Get* – ceea ce vezi este anume ceea ce vei obține). În acest regim, dezvoltatorul Web operează în mod direct doar cu obiectele paginilor Web, fără a apela la posibilitățile oferite de limbajul HTML.

Amintim că limbajul HTML (*Hyper Text Markup Language* – Limbaj de Marcare a Hipertextului) a fost special inventat pentru a formata textele paginilor Web, a insera în ele elemente de control şi referințe la cele mai diverse obiecte (alte pagini Web, imagini, secvențe audio, secvențe video etc.), aflate atât pe calculatorul local, cât şi oriunde pe Internet.

Deşi lucrul în regimul WYSIWYG este cu mult mai comod pentru dezvoltatorii de pagini Web decât cel de marcare a textelor cu ajutorul limbajului HTML, în multe cazuri „scrierea" nemijlocită a paginilor Web în limbajul HTML asigură o economie semnificativă de memorie şi oferă designerului mai multe posibilități de aşezare în pagină şi de formatare a fiecăruia dintre obiectele din componența acesteia.

Se recomandă ca studierea limbajului HTML să se facă pe etape:

1. Elaborarea documentelor Web simple. La această etapă o atenție deosebită trebuie acordată înțelegerii conceptului limbajului HTML, rolului marcatorilor (etichetelor), modalităților de inserare a acestora în textele simple, neformatate, ce sunt afişate de programele de navigare ca pagini Web. Se va parcurge lista de marcatori, se va clarifica destinația fiecăruia dintre ei.

Crearea paginilor Web propriu-zise va începe cu elaborarea structurii generale a acestora, stabilirea obiectelor ce vor fi inserate în ele. La această etapă vor fi studiate în profunzime doar instrumentele HTML destinate formatării textelor:

– titluri;
– paragrafe;
– comentarii;
– stiluri fizice;
– stiluri logice;
– separatorii;
– liste ordonate;
– liste neordonate;
– liste de definiții.

Studierea instrumentelor de formatare a textelor din componența documentelor HTML se va face în strânsă legătură cu materiile referitoare la procesarea textelor studiate în clasele gimnaziale. Totodată, o atenție deosebită se va acorda cuvintelor-cheie din componența etichetelor, modalităților de organizare a etichetelor

în perechi, în cele de început şi cele de sfârşit. Întrucât repertoriul de atribute ale etichetelor este unul foarte vast, eforturile de învățare vor fi concentrate nu pe memorarea acestora, ci pe formarea abilităților de utilizare a sistemelor de asistență *online* şi *offline*, ce pot furniza informații detaliate despre fiecare etichetă şi despre fiecare dintre atributele acesteia.

Activitățile practice recomandate pentru această etapă includ crearea paginilor Web care, pentru început, conțin doar texte formatate în aşa mod, încât să pună în evidență mesajele de transmis:

– Casa mea / Şcoala mea / Oraşul meu / Satul natal;
– Clasa mea / Cercul meu şcolar / Secția mea sportivă;
– Consiliul elevilor / Consiliul tineretului;
– Voluntariatul din şcoala mea, din satul / oraşul meu;
– Librăria / Biblioteca / Casa de cultură / Salonul de muzică;
– Muzeul satului / oraşului meu;
– Istoria satului / oraşului meu;
– Oamenii remarcabili din satul / oraşul meu;
– Arta în viața mea (pe domenii: muzica, arta plastică, arta decorativă);
– Sportul în viața mea;
– E minunat să fii sănătos;
– Săli de sport / de fitness;
– Saloane de înfrumusețare / Saloane de modă;
– Ateliere de cusut / de reparații (auto, calculatoare, electrocasnice, audio, video);
– Gospodăria țărănească;
– Magazine (mărfuri pentru copii, mărfuri şcolare, de uz casnic, îmbrăcăminte).

Dacă în clasele precedente elevii au elaborat deja astfel de pagini, se recomandă afişarea fişierelor HTML în regimul *View Source* (Vezi Sursa) şi analiza modalităților de utilizare a marcatorilor (etichetelor) de formatare a textelor din componența acestora.

2. Crearea legăturilor. Cel mai răspândit serviciu al Internetului – WWW (*World Wide Web* – Pânza Mondială [de Păianjen]), se bazează pe documente HTML, legate între ele, şi fişiere, externe în raport cu documentele respective. Trecerea de la un document HTML la altul şi afişarea/redarea de către programul de navigare a fişierelor externe se realizează cu ajutorul unor legături (referințe) speciale, denumite în ultimul timp şi *linkuri*.

În cadrul unui document HTML, legăturile se creează asociind zonelor active ale paginii adrese URL. În calitate de zone active pot fi fragmentele de text sau de imagini, iar executarea unui clic pe o astfel de zonă impune programul de navigare să găsească resursa asociată pe calculatorul local sau în spațiul virtual, după caz, şi să o afişeze/să o deruleze pe calculatorul local.

Se recomandă ca procesul de studiere a modalităților de creare a legăturilor să urmeze paşii:

– familiarizarea cu terminologia utilizată: ancoră, adresă URL, legătură internă, legătură externă, cale, comentariu la legătură;
– dezvoltarea capacităților de creare/de selectare a zonelor active ale documentelor HTML, formate atât din fragmente de text, cât şi din imagini;
– finalizarea cu formatul marcatoarelor destinate creării legăturilor;

– crearea legăturilor către un document extern, aflat în acelaşi sau în alt catalog de pe calculatorul local;

– crearea legăturilor către un site;

– crearea legăturilor către o secvență a aceluiaşi sau a altui document HTML;

– crearea facilităților de lansare în execuție a unei aplicații de expediere a mesajelor;

– crearea facilităților de copiere a fişierelor.

La etapa inițială, în calitate de documente HTML în care vor fi create legături, se vor utiliza fişierele propuse de profesor, iar după formarea abilităților necesare se va trece la dezvoltarea acestora prin crearea de legături în documentele HTML, elaborate de elevi în cadrul proiectelor listate în activitățile practice ale etapei precedente (Casa mea, Clasa mea, Consiliul elevilor, Voluntariatul ş.a.).

3. Inserarea obiectelor multimedia. Inițial, paginile HTML, afişate pe ecran, conțineau doar texte. Ulterior, odată cu dezvoltarea tehnologiilor informației şi a comunicațiilor, în ele au început să fie inserate legături la cele mai diverse obiecte multimedia: imagini statice, imagini animate, fotografii, secvențe sonore, secvențe video ş.a.

Procesul de studiere a modalităților de inserare în documentele HTML a legăturilor către obiectele multimedia va urma paşii:

– familiarizarea cu formatele fişierelor grafice, audio şi video;

– organizarea fişierelor ce conțin obiecte multimedia în directoare, asociate cu documentele HTML în curs de elaborare;

– familiarizarea cu proprietățile obiectelor multimedia, relevante pentru inserarea lor în documentele HTML: dimensiuni, chenare, comentarii, controale, legături;

– studierea specificului obiectelor multimedia integrate: secvențe de control, atribute de integrare;

– inserarea în documentele HTML a imaginilor, secvențelor sonore sau video;

– setarea proprietăților obiectelor multimedia, inserate în documentele HTML.

La etapa inițială, formarea abilităților practice de inserare a obiectelor multimedia în documentele HTML se va realiza utilizând fişierele HTML şi obiectele multimedia propuse de profesor. Ulterior, dezvoltarea acestor abilități se va realiza în cadrul proiectelor elaborate de elevi, tematica orientativă a cărora este indicată mai sus, în lista respectivă din prima etapă. În acest scop, elevii vor pregăti de sine stătător, în mod individual sau în echipă, acele obiecte multimedia (imagini, fotografii, secvențe sonore, secvențe video) ce contribuie semnificativ la formarea şi punerea în evidență a mesajelor preconizate.

În procesul creării şi/sau selectării obiectelor multimedia ce vor fi incluse în componența documentelor HTML se vor respecta cu strictețe etica digitală şi dreptul de autor, în niciun caz nu se va recurge la plagiat.

4. Organizarea conținuturilor. Versiunile moderne ale limbajului HTML oferă mai multe facultăți de organizare a conținuturilor. În cazul învățământului liceal vom studia doar organizarea conținuturilor cu ajutorul tabelelor. În acest scop, recomandăm elevilor:

– să-şi împrospăteze cunoştinţele despre tabele: titlu, rând, coloană, celulă, titlu de rând, titlu de coloană, proprietăţile celulelor, proprietăţile conţinuturilor din celule;

– să se familiarizeze cu marcatorii utilizaţi pentru crearea şi formatarea tabelelor şi cu atributele acestora;

– să organizeze conţinuturile paginilor Web în curs de elaborare, utilizând aşezarea în pagină cu ajutorul tabelelor;

– să ofere paginilor Web un design ce ar pune în evidenţă mesajele de transmis, editând şi fremătând tabelele respective: combinarea şi divizarea celulelor, aplicarea efectelor de culoare şi de gradient, setarea proprietăţilor chenarelor şi ale bordurilor.

Ca şi în cazul etapelor precedente, iniţial, marcatorii de tabel şi atributele acestora vor fi studiaţi utilizând documente HTML foarte simple, propuse de profesor sau create de sine stătător de elevi, iar ulterior – în cadrul proiectelor elaborate de elevi în mod individual sau în echipă.

5. Testarea documentelor HTML. Pentru început, documentul HTML va fi testat local cu ajutorul unui program de navigare. În procesul testării se va verifica accesibilitatea tuturor obiectelor din componenţa documentelor HTML, corectitudinea referinţelor, funcţionarea instrumentelor de navigare, corectitudinea afişării imaginilor, corectitudinea redării secvenţelor audio şi video.

În cazul depistării unor erori, se va reveni la etapa de creare a paginilor Web şi a obiectelor din componenţa acestora.

La finele studierii acestui modul, elevii sunt invitaţi să facă o analiză colectivă a documentelor HTML elaborate de ei, utilizând în acest scop atât criterii tehnice (corpurile de literă utilizate şi mărimea acestora, aşezarea în pagină, calitatea imaginilor, fidelitatea redării secvenţelor audio şi video, comoditatea navigării), cât şi artistice (originalitatea, relevanţa conţinuturilor în raport cu mesajele de transmis, modul de punere în evidenţă a mesajelor, asortarea culorilor şi a stilurilor de afişare).