



```
Program P151;
{ Suma cifrelor unui număr natural }
type Natural=0..MaxInt;
var i, K, m, n : Natural;

function SumaCifrelor(i:Natural):Natural;
var suma : Natural;
begin
  suma:=0;
  repeat
    suma:=suma+(i mod 10);
    i:=i div 10;
  until i=0;
  SumaCifrelor:=suma;
end; { SumaCifrelor }
```

```
// Program P151
// Suma cifrelor unui număr natural
#include <iostream>
using namespace std;
int i, K = 0, m, n;

int SumaCifrelor(int i)
{
  if (i == 0) return 0;
  else return (i % 10) + SumaCifrelor(i / 10);
} // SumaCifrelor

bool SolutiePosibila(int i)
{
  if (SumaCifrelor(i)== m) return true;
  else return false;
} // SolutiePosibila

void PrelucrareaSolutiei(int i)
```

ANATOL GREMALSCHI  
SERGIU CORLAT  
ANDREI BRAICOV

# Informatică

Manual pentru clasa a XII-a

```
for i:=0 to n do
  if SolutiePosibila(i) then PrelucrareaSolutiei(i);
  writeln('K=', K);
  readln;
end.
```



MINISTERUL EDUCAȚIEI ȘI CERCETĂRII AL REPUBLICII MOLDOVA

---

ANATOL GREMALSCHI  
SERGIU CORLAT  
ANDREI BRAICOV

---



# Informatică

Manual pentru clasa a XII -a

Acumul este proprietate publică, editat din sursele financiare ale Fondului special pentru manuale.

Manualul școlar a fost elaborat în conformitate cu prevederile Curriculumului școlar disciplină, aprobat prin Ordinul ministrului educației și cercetării nr. 906, din 17 iulie 2019.

Manualul a fost aprobat prin Ordinul ministrului educației și cercetării nr. 1928, din 27.12.2024, ca urmare a evaluării calității metodico-științifice.

---

(Denumirea instituției de învățământ)

### EVIDENȚA UTILIZĂRII MANUALULUI:

Anul de folosire	Numele și prenumele elevului	Anul de studii	Aspectul manualului	
			La primire	La returnare

Dirigintele clasei verifică dacă numele, prenumele elevului sunt scrise corect.

Elevii nu vor face niciun fel de însemnări în manual.

Aspectul manualului (la primire și la returnare) se va aprecia de diriginte cu unul dintre calificativele: nou, bun, satisfăcător, nesatisfăcător.

ÎNTreprinderea  
EDITORIAL-POLIGRAFICĂ  
**ȘTIINȚA**

str. Academiei, nr. 3; MD-2028, Chișinău, Republica Moldova  
tel.: +373 22 739616 (anticamera)  
+373 22 739983; +373 60966868 (vânzări)  
e-mail: prini\_știinta@yahoo.com  
www.editurastiinta.md

**Responsabilă de ediție:** Natalia Ciobanu

**Redactor:** Mircea V. Ciobanu

**Corectori:** Mariana Belenciu, Maria Cornescu

**Redactor tehnic:** Nina Duduciu

**Machetare computerizată și copertă:** Romeo řveț

Î.E.P. Știința se obligă să achite deținătorilor de copyright, care încă nu au fost contactați, costurile de reproducere a imaginilor folosite în prezenta ediție.

Toate drepturile asupra acestei ediții aparțin Întreprinderii Editorial-Poligrafice Știința.

#### Descrierea CIP a Camerei Naționale a Cărții

Gremalschi, Anatol.

**Informatică:** Manual pentru clasa a 12-a/Anatol Gremalschi, Sergiu Corlat, Andrei Braicov; Ministerul Educației și Cercetării al Republicii Moldova. – [Chișinău]: Știința, 2025 (Bavat-Print). – 200 p.: fig., tab. color.

Editat din sursele financiare ale Fondului special pentru manuale.

ISBN 978-9975-85-492-4.

## Dragi prieteni!

Informatica ca știință și Tehnologia informației și comunicațiilor ca un ansamblu de metode și mijloace bazate pe realizările acestei științe sunt în continuă dezvoltare. De acest fapt ne putem convinge în baza propriei experiențe: în fiecare an apar noi echipamente și dispozitive digitale, noi produse-program și noi servicii online. Practica cotidiană ne demonstrează încă o dată faptul că mijloacele tehnologiei informației și comunicațiilor penetreză toate domeniile vieții economice și sociale, contribuind semnificativ la creșterea productivității muncii, modificarea modului în care muncim, învățăm, ne odihnim, comunicăm cu alte persoane, difuzăm informații despre noi și aflăm informații despre alții, ne realizăm potențialul creativ.

Ca să nu ne pierdem în această diversitate de echipamente și produse digitale, ca să utilizăm efectiv și eficient produsele-program, ca să putem accesa în siguranță serviciile digitale, este foarte important să avem o înțelegere profundă a principiilor care stau la baza informaticii și a tehnologiei informației și comunicațiilor. De asemenea, este important să ne formăm și să ne dezvoltăm abilitățile de utilizare a acestor tehnologii pentru a transmite, receptiona, stoca și prelucra informații numerice, textuale, audio și video, pentru a crea cele mai diverse produse digitale: programe de calculator, baze de date, site-uri și pagini Web, creații multimedia.

Prin urmare, indiferent de profilul de liceu pe care îl urmați, umanist sau real, sportiv sau artistic, pentru a vă atinge scopurile de dezvoltare personală, inclusiv pentru o carieră profesională de succes și o viață împlinită, neapărat veți avea nevoie de competențe digitale, formarea și dezvoltarea cărora reprezintă scopul acestui manual. Mai exact, la finele clasei a XII-a veți putea:

- să elaborați și să implementați subalgoritmii destinați soluționării problemelor frecvent întâlnite în viața cotidiană;
- să selectați și să implementați pe calculator cele mai răspândite tehnici de programare;
- să elaborați și să implementați pe calculator modelele informatic ale celor mai diverse obiecte, să aplicați metodele frecvent utilizate de calcul numeric;
- să organizați și să prelucrați informațiile cu ajutorul sistemelor de gestiune a bazelor de date.

De asemenea, studierea materiilor incluse în manual va contribui la formarea și consolidarea următoarelor aptitudini și valori:

- preocupare pentru cunoașterea sinelui și a lumii prin mijloacele digitale;
- corectitudine și coerentă în utilizarea mijloacelor digitale, inițiativă și perseverență în algoritmizarea problemelor și implementarea algoritmilor;
- curiozitate și interes, atitudine critică și creativă în demersul de cunoaștere a lumii cu ajutorul modelărilor pe calculator;
- respectarea regulilor de securitate, ergonomice, etice și de design în crearea și difuzarea produselor digitale.

Studierea unuia dintre modulele la alegere, descrise în Capitolul 5, va contribui la valorificarea metodelor și a instrumentelor specifice prelucrărilor digitale, la dezvoltarea gândirii și a creativității în integrarea achizițiilor informatic cele din alte domenii. Extensiile din Capitolul 6, care sunt optionale, sunt destinate studierii aprofundată a unor teme din Informatică.

Se recomandă ca modulele la alegere și extensiile să fie studiate prin metoda clasei inverse. Această metodă se bazează pe studiul de sine stătător al elevilor, profesorului revenindu-i rolul de consultant și îndrumător.

Principalele resurse educaționale recomandate elevilor pentru studierea modulelor la alegere și a extensiilor sunt parte componentă a acestui manual și pot fi descărcate de pe pagina Web <http://www.ctice.gov.md> a Centrului Tehnologii Informaționale și Comunicaționale în Educație.

Vă dorim succese,  
Autorii

# CUPRINS

1

## Funcții și proceduri

1.1.	Subprograme .....	6
1.2.	Functii .....	8
1.3.	Proceduri / Funcții fără tip.....	15
1.4.	Domenii de vizibilitate (PASCAL) .....	23
1.5.	Domenii de vizibilitate (C/C++) .....	26
1.6.	Comunicarea prin variabile globale .....	29
1.7.	Efecte colaterale .....	33
1.8.	Recursia .....	40
1.9.	Sintaxa declarațiilor și apelurilor de subprograme .....	44

2

## Tehnici de programare

2.1.	Complexitatea algoritmilor .....	48
2.2.	Estimarea necesarului de memorie .....	50
2.3.	Măsurarea timpului de execuție .....	55
2.4.	Estimarea timpului cerut de algoritm .....	61
2.5.	Complexitatea temporală a algoritmilor .....	67
2.6.	Iterativitate sau recursivitate .....	70
2.7.	Metoda trierii .....	75
2.8.	Tehnica Greedy .....	81

3

## Modelare și calcul numeric

3.1.	Noțiune de model. Clasificarea modelelor .....	87
3.2.	Modelul matematic și modelarea matematică .....	89
3.3.	Soluții analitice și soluții de simulare .....	92
3.4.	Etapele rezolvării problemei la calculator .....	95
3.5.	Numere aproximative. Eroarea absolută și eroarea relativă .....	98
3.6.	Sursele erorilor de calcul .....	100
3.7.	Separarea soluțiilor ecuațiilor algebrice și transcendente .....	103
3.8.	Metoda bisecției .....	105
3.9.	Metoda coardelor .....	109
3.10.	Metoda dreptunghiurilor pentru calculul aproximativ al integralei definite .....	113
3.11.	Variatii ale metodei dreptunghiurilor .....	118

**4****Baze de date**

4.1.	Date și baze de date .....	123
4.2.	Tipuri de baze de date .....	125
4.3.	Elaborarea bazelor relaționale de date .....	128
4.4.	Sisteme de gestiune a bazelor de date .....	132
4.5.	Crearea tabelelor .....	135
4.6.	Stabilirea relațiilor dintre tabele .....	141
4.7.	Modificarea tabelelor .....	143
4.8.	Expresii Access .....	147
4.9.	Noțiuni generale despre interogări .....	152
4.10.	Interrogări de selecție .....	156
4.11.	Interrogări de acțiune .....	160
4.12.	Interrogări de totalizare .....	163
4.13.	Formulare .....	167
4.14.	Rapoarte .....	177
4.15.	Mentenanța bazelor de date .....	183

**5****Module la alegere**

5.1.	Prelucrări avansate ale informațiilor din bazele de date în formă de liste .....	186
5.2.	Metode experimentale în științele umanistice .....	187
5.3.	Programarea Web .....	190
5.4.	Structuri dinamice de date în PASCAL .....	192
5.5.	Structuri dinamice de date în C/C++ .....	195

**6 Extensii .....** 197

# Functii și proceduri

## 1.1 Subprograme

E cunoscut faptul că o problemă complexă poate fi rezolvată prin divizarea ei într-un set de părți mai mici (subprobleme). Pentru fiecare parte se scrie o anumită secvență de instrucțiuni, denumită **subprogram**. Problema în ansamblu se rezolvă cu ajutorul programului principal, în care pentru rezolvarea subproblemelor se folosesc apelurile subprogramelor respective. Când în programul principal se întâlnește un apel, execuția continuă cu prima instrucțiune din programul apelat (fig. 1.1). Când se termină executarea instrucțiunilor din subprogram, se revine la instrucțiunea imediat următoare apelului din programul principal.

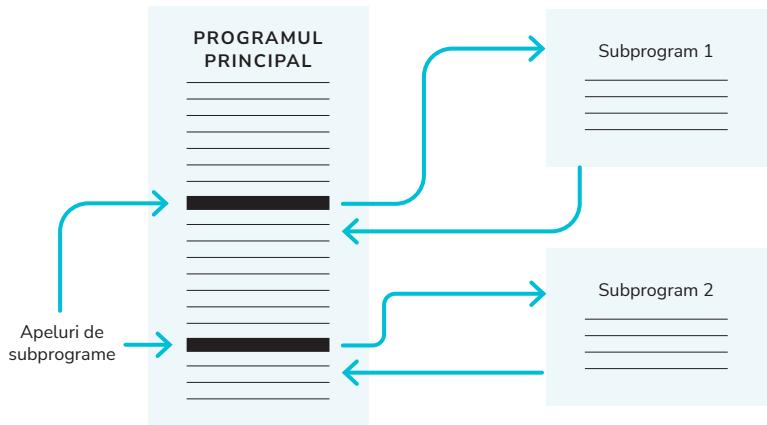


Fig. 1.1. Interacțiunea dintre program și subprogram

### Pascal

În limbajul PASCAL există două tipuri de subprograme, și anume funcții și proceduri:

```
<Subprograme> ::= { <Funcție>; | <Procedură>; }
```

**Funcțiile** sunt subprograme care calculează și returnează o valoare. Limbajul PASCAL conține un set de funcții predefinite, cunoscute oricărui program: `sin`, `cos`, `eof` etc. În completare, programatorul poate defini funcții proprii, care se apelează în același mod ca funcțiile-standard. Prin urmare, conceptul de funcție extinde noțiunea de expresie PASCAL.

**Procedurile** sunt subprograme care efectuează prelucrarea datelor comunicate în momentul apelului. Limbajul conține procedurile predefinite `read`, `readln`, `write`, `writeln` și.a., studiate în clasele precedente. În completare, programatorul poate defini proceduri proprii, care se apelează în același mod ca procedurile-standard. Prin urmare, conceptul de procedură extinde noțiunea de instrucțiune PASCAL.

## C++

În limbajele C/C++ subprogramele poartă denumiri de funcții. Limbajele C/C++ conțin seturi de funcții predefinite, grupate în librării. Funcțiile unei librării devin cunoscute programului în curs de elaborare prin includerea în el a librăriei respective. Amintim că includerea unei librării se realizează cu ajutorul directivei `#include`:

```
#include <Nume librărie>
```

De exemplu, pentru a putea utiliza funcțiile de citire/scriere, programul în curs de elaborare trebuie să conțină directiva:

```
#include <iostream>
```

Pentru a utiliza funcțiile matematice, programul în curs de elaborare trebuie să conțină directiva:

```
#include <math.h>
```

Pe lângă funcțiile predefinite, grupate în librării, programatorul poate defini și funcțiile proprii. Funcțiile definite de programator se apelează exact în același mod ca și funcțiile predefinite.

În limbajele C/C++ deosebim două categorii de funcții:

**1) Funcții cu tip.** Apelul unor astfel de funcții se realizează prin includerea denumirii acesteia în orice expresie C/C++. Fiind apelată, funcția calculează și returnează o valoare, tipul căreia este indicat în mod explicit în declarația funcției.

Exemple: `sin`, `cos`, `abs`, `fabs` etc.

**2) Funcții fără tip (`void`).** Apelul unei funcții fără tip se realizează prin includerea denumirii acesteia în orice loc din program în care poate apărea o instrucțiune. Fiind apelată, funcția prelucrează datele ce-i sunt comunicate în momentul apelului.

Exemple: `scanf`, `printf`, `eoln` și.a.

Evident, conceptul de *funcție cu tip* extinde noțiunea de **expresie**, iar conceptul *funcție fără tip* – noțiunea de **instrucțiune**.

## Pascal C++

Indiferent de limbajul de programare utilizat, subprogramele se definesc, în întregime, în partea declarativă a unui program. Evident, apelurile de funcții și proceduri se includ în partea executabilă a programului.

Un subprogram poate fi apelat chiar de el însuși, caz în care apelul este **recursiv**.

## Întrebări și exerciții

- 1 Explicați termenii program principal și subprogram.
- 2 Cum interacționează programul și subprogramul?
- 3 (PASCAL) Care este diferența dintre proceduri și funcții?
- 4 (C/C++) Care este diferența dintre funcții cu tip și funcții fără tip?
- 5 **Sistematizează-ți cunoștințele!** (PASCAL) Cum se apelează o funcție? În care locuri ale programului pot apărea apeluri de funcții? Cum se apelează o procedură? În care locuri ale programului pot apărea apeluri de proceduri?
- 6 **Sistematizează-ți cunoștințele!** (C/C++) Cum se apelează o funcție cu tip? În care locuri ale programului pot apărea apeluri de funcții cu tip? Cum se apelează o funcție fără tip? În care locuri ale programului pot apărea apeluri de funcții fără tip?
- 7 **Observă!** (PASCAL) Numiți tipul argumentului și tipul rezultatului furnizat de funcțiile predefinite `abs`, `chr`, `eof`, `eoln`, `exp`, `ord`, `sin`, `sqr`, `sqrt`, `pred`, `succ`, `trunc`. Numiți tipul parametrilor actuali ai procedurilor `read` și `write`.
- 8 **Observă!** (C/C++) Numiți tipul rezultatului furnizat de funcțiile cu tip, incluse în librăria `<math.h>`: `abs`, `fabs`, `exp`, `sin`, `sqr`, `sqrt`, `trunc`.
- 9 **Cercetează!** (PASCAL) Ce prelucrări de date efectuează procedurile `read` și `write`?
- 10 **Cercetează!** (C/C++) Ce prelucrări de date efectuează funcțiile fără tip `scanf` și `printf` ale limbajului C sau echivalentele lor `cin` și `cout` în C++?

## 1.2 Funcții

### Pascal

Textul PASCAL al unei **declarații de funcție** are forma:

```
function f(x1, x2, ..., xn) : tr ;
D;
begin
  ...
  f := e;
  ...
end;
```

Prima linie este antetul funcției, format din:

$f$  – numele funcției;

$(x_1, x_2, \dots, x_n)$  – lista optională de parametri formali reprezentând argumentele funcției;

$t_r$  – tipul rezultatului; acesta trebuie să fie numele unui tip simplu sau tip referință.

Antetul este urmat de **corpul funcției** format din declarațiile locale optionale  $D$  și instrucțiunea compusă `begin ... end`.

Declarațiile locale sunt grupate în secțiunile (eventual vide) **label**, **const**, **type**, **var**, **function/procedure**.

Numele  $f$  al funcției apare cel puțin o dată în partea stângă a unei instrucțiuni de atribuire care se execută:  $f := e$ . Ultima valoare atribuită lui  $f$  va fi întoarsă în programul principal.

În mod obișnuit, un parametru formal din lista  $(x_1, x_2, \dots, x_n)$  are forma:

$v_1, v_2, \dots, v_k : t_p$

unde  $v_1, v_2, \dots, v_k$  sunt identificatori, iar  $t_p$  este un nume de tip.

Utilizarea funcției  $f$  se specifică printr-un apel de forma:

$f(a_1, a_2, \dots, a_n)$

unde  $(a_1, a_2, \dots, a_n)$  este **lista de parametri actuali**. De obicei, parametrii actuali sunt expresii, valorile cărora sunt comunicate funcției. Corespondența dintre un parametru actual și parametrul formal se face prin poziția ocupată de acestia în cele două liste. Parametrul actual trebuie să fie **compatibil** din punctul de vedere al atribuirii cu tipul parametrului formal.

**Exemplu:**

### Pascal

```
Program P97;
{Declararea si utilizarea functiei Putere }
type Natural=0..MaxInt;
var a : real;
    b : Natural;
    c : real;
    s : integer;
    t : integer;
    v : real;
function Putere(x : real; n : Natural) : real;
    {calcularea lui x la puterea n }
var p : real;
    i : integer;
begin
  p:=1;
  for i:=1 to n do p:=p*x;
  Putere:=p;
end; { Putere }
begin
  a:=3.0;
  b:=2;
  c:=Putere(a, b);
  writeln(a:10:5, b:4, c:10:5);
  s:=2;
  t:=4;
  v:=Putere(s, t);
  writeln(s:5, t:4, v:10:5);
  readln;
end.
```

Funcția Putere are doi parametri formali:  $x$  de tipul **real** și  $n$  de tipul **Natural**. Funcția returnează o valoare de tipul **real**. În corpul funcției sunt declarate variabilele locale  $p$  și  $i$ .

La execuția apelului  $\text{Putere}(a, b)$  valorile 3.0 și 2 ale parametrilor actuali  $a, b$  se transmit parametrilor formali  $x$  și, respectiv,  $n$ . De menționat că tipul lui  $a$  coincide cu tipul lui  $x$  și tipul lui  $b$  coincide cu tipul lui  $n$ .

În cazul apelului  $\text{Putere}(s, t)$ , tipul parametrilor actuali  $s, t$  nu coincide cu tipul parametrilor formali  $x$  și, respectiv,  $n$ . Totuși, apelul este corect, întrucât tipurile respective sunt compatibile din punctul de vedere al atribuirii.

Textul C/C++ al unei **declarații de funcție** are forma:

### C++

```
<Tip rezultat> <Nume functie> (x1, x2, ..., xn)
D;
{
    ...
    return <Rezultat>
}
```

Prima linie este antetul funcției, format din:

<Tip rezultat> – tipul rezultatului ce va fi returnat. Acesta trebuie să fie numele unui tip simplu sau tip referință.

<Nume funcție> – numele funcției.

(x<sub>1</sub>, x<sub>2</sub>, ..., x<sub>n</sub>) – lista optională de parametri formali reprezentând argumentele funcției.

Antetul este urmat de **corful funcției** format din declarațiile locale optionale D și instrucțiunea compusă { ... }.

Declarațiile locale sunt preponderent declarații de variabile. Variabilele, declarate în corpul funcției sunt numite **variabile locale**.

Numele funcției este folosit pentru apelul acesteia. Fiind apelată, funcția va returna în programul apelant valoarea <Rezultat>, indicată după cuvântul-cheie **return**.

În mod obișnuit, un parametru formal din lista (x<sub>1</sub>, x<sub>2</sub>, ..., x<sub>n</sub>) are forma:

```
t1 v1, t2 v2, ..., tn vn
```

unde perechile t<sub>1</sub> v<sub>1</sub>, t<sub>2</sub> v<sub>2</sub>, ..., t<sub>n</sub> v<sub>n</sub> sunt formate din <Nume tip> și <Identificator>.

Un apel de funcție are forma:

```
<Nume functie> (a1, a2, ..., an)
```

unde (a<sub>1</sub>, a<sub>2</sub>, ..., a<sub>n</sub>) este **lista de parametri actuali**. De obicei, parametrii actuali sunt expresii, valorile cărora sunt comunicate funcției. Corespondența dintre un parametru actual și parametrul formal se face prin poziția ocupată de acestia în cele două liste. Parametrul actual trebuie să fie **compatibil** din punctul de vedere al atribuirii cu tipul parametrului formal.

**Exemplu:**

### C++

```
// Program P97;
// Declararea si utilizarea functiei Putere

#include <iostream>
using namespace std;

double putere (double x, int n)
{
    double p = 1.0;
    for (int i = 1; i <= n; i++)
        p = p * x;
```

```

    return p;
}

int main()
{
    double a = 3.0, c;
    int b = 2;
    c = putere(a, b);
    cout << a << "^" << b << " = " << c << endl;
    int s = 2, t = 4;
    c = putere(s, t);
    cout << s << "^" << t << " = " << c << endl;
    return 0;
}

```

Funcția `putere` are doi parametri formali: `x` de tipul real cu precizie dublă și `n` de tipul `int`. Funcția returnează o valoare de tip real, cu precizie dublă. În corpul funcției sunt declarate variabilele locale `p` și `i`.

La execuția apelului `putere(a, b)`, valorile 3.0 și 2 ale parametrilor actuali `a`, `b` se transmit parametrilor formali `x` și, respectiv, `n`. De menționat că tipul lui `a` coincide cu tipul lui `x` și tipul lui `b` coincide cu tipul lui `n`.

În cazul apelului `putere(s, t)`, tipurile parametrilor actuali `s`, `t` nu coincid cu tipurile parametrilor formali `x` și, respectiv, `n`. Totuși, apelul este corect, întrucât tipurile respective sunt compatibile din punctul de vedere al atribuirii.

## Întrebări și exerciții

1

**Aplică!**

Se consideră următoarea declarație:

**P**ascal

```

function Factorial(n : integer) : integer;
var p, i : integer;
begin
    p:=1;
    for i:=1 to n do p:=p * i;
    Factorial:=p;
end;

```

**C**++

```

int Factorial(int n)
// calcularea n!
{
    int p = 1;
    for (int i = 1; i <= n; i++) p = p * i;
    return p;
} // sfarsit Factorial

```

Numiți tipul parametrului formal și tipul rezultatului returnat de funcție. Precizați variabilele declarate în corpul funcției. Elaborați un program care afișează la ecran valorile  $n!$  pentru  $n = 2, 3$  și  $7$ .

2 În care loc al programului principal se includ declarațiile de funcții?

3 **Observă!** Comentați erorile de compilare a programului ce urmează:

### Pascal

```
Program P98;
  { Eroare }
function Factorial(n : 0..7) : integer;
var p, i : integer;
begin
  p:=1;
  for i:=1 to n do p:=p*i;
  Factorial:=p;
end; { Factorial }
begin
  writeln(Factorial(4));
  readln;
end.
```

### C++

```
// Program P98
// Eroare

#include <iostream>
using namespace std;
enum par {v1 = 1, v2 = 2, v3 = 3, v4 = 4, v5 = 5, v6 = 6, v7 = 7};

int factorial(enum par n)
{
    int p = 1;
    for (int i = 1; i <= n; i++) p = p * i;
    return p;
}

int main()
{
    cout << factorial(4) << endl;
    return 0;
}
```

4 **Exersează!** Se consideră antetul:

### Pascal

```
function F(x : real; y : integer; z : char) : boolean;
```

### C++

```
bool F(double x, int y, char z)
```

Care dintre apelurile ce urmează sunt corecte:

- |    |                   |    |                     |
|----|-------------------|----|---------------------|
| a) | $F(3.18, 4, 'a')$ | e) | $F(3.18, 4, 4)$     |
| b) | $F(4, 4, '4')$    | f) | $F('3.18', 4, '4')$ |
| c) | $F(4, 4, 4)$      | g) | $F(15, 21, '3')$    |
| d) | $F(4, 3.18, 'a')$ | h) | $F(15, 21, 3)$      |

**5 Programează!** Elaborați o funcție care calculează:

- a) suma numerelor reale  $a, b, c, d$ ;
- b) media numerelor întregi  $i, j, k, m$ ;
- c) minimumul din numerele  $a, b, c, d$ ;
- d) numărul de vocale într-un sir de caractere;
- e) numărul de consoane într-un sir de caractere;
- f) rădăcina ecuației  $ax + b = 0$ ;
- g) cel mai mic divizor al numărului întreg  $n > 0$ , diferit de 1;
- h) cel mai mare divizor comun al numerelor naturale  $a, b$ ;
- i) cel mai mic multiplu comun al numerelor naturale  $a, b$ ;
- j) ultima cifră în notația zecimală a numărului întreg  $n > 0$ ;
- k) câte cifre sunt în notația zecimală a numărului întreg  $n > 0$ ;
- l) cifra superioară în notația zecimală a numărului întreg  $n > 0$ ;
- m) numărul de aparitii ale caracterului dat într-un sir de caractere.

**6 Programează!** Se consideră următoarele declarații:

**Pascal**

```
const nmax=100;
type Vector=array [1..nmax] of real;
```

**C++**

```
#define nmax 100
double Vector[nmax];
```

Elaborați o funcție care calculează:

- a) suma componentelor unui vector;
- b) media componentelor vectorului;
- c) componenta maximă;
- d) componenta minimă.

7

**Analizează și aplică!**

Se consideră următoarele tipuri de date:

**Pascal**

```
type Punct=record
    x, y : real
  end;
Segment=record
    A, B : Punct
  end;
Triunghi=record
    A, B, C : Punct
  end;
Dreptunghi=record
    A, B, C, D : Punct
  end;
Cerc=record
    Centru : Punct;
    Raza : real
  end;
```

**C++**

```
struct Punct {double x; double y;}
struct Segment {struct Punct A; struct Punct B;}
struct Triunghi {struct Segment A; struct Segment B, struct Segment C;}
struct Dreptunghi
{
    struct Segment A; struct Segment B;
    struct Segment C; struct Segment D;
}
struct Cerc { struct Punct; double Raza;}
```

Elaborați o funcție care calculează:

- a) lungimea segmentului;
- b) lungimea cercului;
- c) aria cercului;
- d) aria triunghiului;
- e) aria dreptunghiului.

8

**Aplică!**

(PASCAL) Variabila A este introdusă prin declarația

```
var A : set of char;
```

Elaborați o funcție care returnează numărul de elemente din mulțimea A.

9

**Aplică!**

(C/C++) Variabila A este introdusă prin declarația

```
string A;
```

Elaborați o funcție care returnează numărul de caractere distințe din sirul de caractere A.

- 10** **Aplică!** Elaborați o funcție care calculează diferența în secunde dintre două momente de timp date prin oră, minute și secunde.
- 11** **Integrează cunoștințele și aplică!** Un triunghi este definit prin coordonatele vârfurilor sale. Scrieți funcții care, pentru două triunghiuri date, verifică dacă:
- triunghiurile au aceeași arie;
  - triunghiurile sunt asemenea;
  - primul triunghi este în interiorul celui de-al doilea triunghi.

## 1.3 Proceduri / Funcții fără tip

### Pascal

Forma generală a textului unei declarații de procedură este:

```
procedure P(x1, x2, ..., xn);
D;
begin
...
end;
```

În **antetul procedurii** apar:

*P* – numele procedurii;  
*x<sub>1</sub>, x<sub>2</sub>, ..., x<sub>n</sub>* – lista optională de parametri formali.

În **corpu procedurii** sunt incluse:

*D* – declarațiile locale (optionale) grupate după aceleași reguli ca în cazul funcțiilor;  
**begin ... end** – instrucțiune compusă; ea nu conține vreo atribuire asupra numelui procedurii.  
Procedura poate să întoarcă mai multe rezultate, dar nu prin numele ei, ci prin variabile desemnate special (cu prefixul **var**) în lista de parametri formali.

Parametrii din listă introdusi prin declarații de forma

*v<sub>1</sub>, v<sub>2</sub>, ..., v<sub>k</sub> : t<sub>p</sub>*

se numesc **parametri-valoare**. Aceștia servesc pentru transmiterea de valori din programul principal în procedură.

Parametrii formali introdusi în listă prin declarații de forma

**var** *v<sub>1</sub>, v<sub>2</sub>, ..., v<sub>k</sub> : t<sub>p</sub>*

se numesc **parametri-variabilă** și servesc pentru întoarcerea rezultatelor din procedură în programul principal.

Activarea unei proceduri  $P$  se face printr-un apel de forma

$P(a_1, a_2, \dots, a_n)$

unde  $a_1, a_2, \dots, a_n$  este lista de **parametri actuali**. Spre deosebire de funcție, apelul de procedură este o instrucțiune; aceasta se inserează în programul principal în locul în care sunt dorite efectele produse de execuția procedurii.

În cazul unui **parametru-valoare**, drept parametru actual poate fi utilizată orice expresie de tipul respectiv, în particular o constantă sau o variabilă. Modificările parametrilor-valoare nu se transmit în exteriorul subprogramului.

În cazul unui **parametru-variabilă**, drept parametri actuali pot fi utilizate numai variabile. Evident, modificările parametrilor în studiu vor fi transmise programului apelant.

**Exemplu:**

### Pascal

```
Program P99;
{Declararea si utilizarea procedurii Lac }
var a, b, c, t, q : real;

procedure Lac(r : real; var l, s : real);
{lungimea si aria cercului }
{r - raza; l - lungimea; s - aria }
const Pi=3.14159;
begin
  l:=2*Pi*r;
  s:=Pi*sqr(r);
end; {Lac }

begin
  a:=1.0;
  Lac(a, b, c);
  writeln(a:10:5, b:10:5, c:10:5);

  Lac(3.0, t, q);
  writeln(3.0:10:5, t:10:5, q:10:5);

  readln;
end.
```

Procedura Lac are trei parametri formali:  $r$ ,  $l$  și  $s$ . Parametrul  $r$  este un parametru-valoare, iar  $l$  și  $s$  sunt parametri-variabilă.

Execuția instrucțiunii  $Lac(a, b, c)$  determină transmiterea valorii 1.0 drept valoare a parametrului formal  $r$  și a locațiilor (adreselor) variabilelor  $b$  și  $c$  drept locații (adrese) ale parametrilor formali  $l$  și  $s$ . Prin urmare, secvența de instrucțiuni

```
a:=1.0;
Lac(a, b, c)
```

este echivalentă cu secvența

```
b:=2*Pi*1.0;
c:=Pi*sqr(1.0)
```

În mod similar, instrucțiunea

```
Lac(3.0, t, q)
```

este echivalentă cu secvența

```
t:=2*Pi*3.0;
q:=Pi*sqr(3.0)
```

## C++

Forma generală a textului unei declarații de funcție fără tip este:

```
void <Nume_funcție> (x1, x2, ..., xn)
{
    D;
    ...
    return;
}
```

În **antetul** funcției fără tip apar:

<Nume\_funcție> – numele funcției;

x<sub>1</sub>, x<sub>2</sub>, ..., x<sub>n</sub> – lista optională de parametri formali.

În **corpu** funcției fără tip sunt incluse:

D – declarațiile locale (optionale) grupate după aceleași reguli ca în cazul funcțiilor cu tip;

{ ... } – instrucțiune compusă; ea nu conține vreo atribuire asupra numelui funcției.

Funcția fără tip poate să întoarcă mai multe rezultate prin variabile desemnate special (prin variabile de tip pointer) în lista de parametri formali.

Parametrii x<sub>1</sub>, x<sub>2</sub>, ..., x<sub>n</sub> din listă sunt introdusi prin declarații de forma t<sub>1</sub> v<sub>1</sub>, t<sub>2</sub> v<sub>2</sub>, ..., t<sub>n</sub> v<sub>n</sub>, unde perechile t<sub>1</sub> v<sub>1</sub>, t<sub>2</sub> v<sub>2</sub>, ..., t<sub>n</sub> v<sub>n</sub> sunt formate din <Nume\_tip> și <Identificator>.

Parametrii t<sub>1</sub> v<sub>1</sub>, t<sub>2</sub> v<sub>2</sub>, ..., t<sub>n</sub> v<sub>n</sub> se numesc **parametri-valoare**. Aceștia servesc pentru transmiterea de valori din programul principal în funcție.

Parametrii formali introdusi în listă prin declarații de forma

```
t1* v1, t2* v2..., tk * vk
```

se numesc **parametri-variabilă** și servesc pentru întoarcerea rezultatelor din funcția curentă în funcția apelantă.

Activarea unei funcții fără tip se face printr-un apel de forma

```
P(a1, a2, ..., an)
```

unde P este numele funcției, iar a<sub>1</sub>, a<sub>2</sub>, ..., a<sub>n</sub> este lista de **parametri actuali**. Spre deosebire de funcție, apelul de funcție fără tip este o instrucțiune; aceasta se inserează în funcția apelantă în locul în care sunt dorite efectele produse de execuția funcției.

În cazul unui **parametru-valoare** drept parametru actual poate fi utilizată orice expresie de tipul respectiv, în particular o constantă sau o variabilă. Modificările parametrilor-valoare nu se transmit în exteriorul subprogramului.

În cazul unui **parametru-variabilă**, drept parametri actuali pot fi utilizate numai variabile. Evident, modificările parametrilor în studiu vor fi transmise funcției apelante.

**Exemplu:**

**C++**

```
// Program P99
// Declararea si utilizarea procedurii Lac

#include <iostream>
#include <math.h>
using namespace std;
double a, b, c, t, q;

void Lac(double r, double* l, double* s)
    //lungimea si aria cercului: r - raza; l - lungimea; s - aria
{
#define Pi 3.14159
    *l = 2 * Pi * r;
    *s = Pi * pow(r, 2);
    return;
}

int main()
{
    a = 1.0;
    Lac(a, &b, &c);
    cout << a << " " << b << " " << c << endl;
    Lac(3.0, &t, &q);
    cout << 3.0 << " " << t << " " << q << endl;
}
```

Funcția Lac are trei parametri formali: r, l și s. Parametrul r este un parametru-valoare, iar l și s sunt parametri-variabilă.

Execuția instrucțiunii Lac(a, b, c) determină transmiterea valorii 1.0 drept valoare a parametrului formal r și a locațiilor (adreselor) variabilelor b și c drept locații (adrese) ale parametrilor formali l și s. Prin urmare, secvența de instrucțiuni

```
a = 1.0;
Lac(a, &b, &c)
```

este echivalentă cu secvența

```
b = 2 * Pi * 1.0;
c = Pi * pow(1.0, 2)
```

În mod similar, instrucțiunea

```
Lac(3.0, &t, &q)
```

este echivalentă cu secvența

```
t = 2 * Pi * 3.0;
q = Pi * pow(3.0, 2)
```

## Întrebări și exerciții

- 1 Observă!** Care este diferența dintre un parametru-valoare și un parametru-variabilă?
- 2 Exersează!** (PASCAL) Se consideră declarațiile:

```
var k, m, n : integer;
      a, b, c : real;
procedure P(i : integer; var j : integer;
            x : real; var y : real);
begin
  {...}
end.
```

Care dintre apelurile ce urmează sunt corecte?

- |    |              |    |             |
|----|--------------|----|-------------|
| a) | P(k,m,a,b)   | f) | P(n,m,6,b)  |
| b) | P(3,m,a,b)   | g) | P(n,m,6,20) |
| c) | P(k,3,a,b)   | h) | P(a,m,b,c)  |
| d) | P(m,m,a,b)   | i) | P(i,i,i,i)  |
| e) | P(m,k,6.1,b) | j) | P(a,a,a,a)  |

Argumentați răspunsul.

- 3 Exersează!** (C/C++) Se consideră declarațiile:

```
int k, m, n;
double a, b, c;

void P(int i, int* j, double x, double* y)
{
  ...
}
```

Care dintre apelurile ce urmează sunt corecte?

- |    |                |    |                 |
|----|----------------|----|-----------------|
| a) | P(k,&m,a,&b)   | f) | P(n,&m, 6, &b)  |
| b) | P(3,&m,a,&b)   | g) | P(n,m,6,20)     |
| c) | P(k,3,a,&b)    | h) | P(a, m, b, c)   |
| d) | P(m,&m,a,&b)   | i) | P(i, &i, i, &i) |
| e) | P(m,&k,6.1,&b) | j) | P(a, &a, a, &a) |

Argumentați răspunsul.

4

**Observă!**

Comentați programul ce urmează:

**Pascal**

```
Program P100;
  {Eroare }
var   a : real;
        b : integer;
procedure P(x : real; var y : integer);
begin
  {... }
end; { P }
begin
  P(a, b);
  P(0.1, a);
  P(1, b);
  P(a, 1);
end.
```

**C++**

```
// Program P100
// Eroare
#include <iostream>
using namespace std;
double a;
int b;
void P(double x, int* y)
{... }

int main()
{
  P(a, &b);
  P(0.1, a);
  P(1, &b);
  P(a, 1);
  return 0;
}
```

5

**Cercetează!**

Ce va afișa la ecran programul ce urmează?

**Pascal**

```
Program P101;
  {Parametru-valoare si parametru-variabila }
var a, b : integer;

procedure P(x : integer; var y : integer);
```

```

begin
    x:=x+1;
    y:=y+1;
    writeln( 'x=', x, ' y=', y);
end; {P }

begin
    a:=0;
    b:=0;
    P(a, b);
    writeln( 'a=', a, ' b=', b);
    readln;
end.

```

### C++

```

// Program P101
// Parametru-valoare si parametru-variabila
#include <iostream>
using namespace std;
int a, b;

void P(int x, int* y)
{
    x = x + 1;
    *y = *y + 1;
    cout <<"x= "<< x<<" y= "<< *y << endl;
    return;
}

int main()
{
    a = 0;
    b = 0;
    P(a, &b);
    cout <<"a= "<< a<<" b= "<< b << endl;
    return 0;
}

```

Argumentați răspunsul.

6

**Programează!** Elaborați o procedură care:

- calculează rădăcinile ecuației  $ax^2 + bx + c = 0$ ;
- radiază dintr-un sir caracterul indicat în apel;
- încadrează un sir de caractere între simbolurile "#";
- ordonează componentele unui tablou ce conține până la 100 de numere reale în ordine crescătoare;
- ordonează componentele unui fișier ce conține numere întregi în ordine descrescătoare;
- calculează și depune într-un tablou numerele prime mai mici decât un număr natural dat  $n$ .

7

**Programează!**

Se consideră următoarele tipuri de date:

**Pascal**

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

**C++**

```
struct Data {
    int Ziua;
    int Luna;
    int Anul;
};

struct Personoana { string NumePrenume;
    struct Data DataNasterii;
} Pers[50];
```

Elaborați o procedură care primește din programul principal o listă de persoane și întoarce:

- persoanele născute în ziua z a lunii;
- persoanele născute în luna  $l$  a anului;
- persoanele născute în anul  $a$ ;
- persoanele născute pe data z.l.a;
- persoana cea mai în vîrstă;
- persoana cea mai tânără;
- vîrsta fiecărei persoane în ani, luni, zile;
- lista persoanelor care au mai mult de  $v$  ani;
- lista persoanelor în ordine alfabetică;
- lista persoanelor ordonată conform datei nașterii;
- lista persoanelor de aceeași vîrstă (născuți în același an).

8

**Integrează cunoștințele și aplică!**

Elaborați o procedură care:

- creează o copie de rezervă a unui fișier text;
- excluză dintr-un fișier text liniile vide;
- numerotează liniile unui fișier text;
- concatenează două fișiere text într-unul singur;
- concatenează  $n$  fișiere text ( $n > 2$ ) într-unul singur.

9

**Integrează cunoștințele și aplică!**

Vom numi *mari* numerele naturale care conțin mai mult de 20 de cifre semnificative. Să se definească un tip de date pentru numerele naturale mari și să se scrie proceduri care să adune și să scadă astfel de numere.

## 1.4

## Domenii de vizibilitate (PASCAL)

Corpul unui program sau subprogram se numește **bloc**. Deoarece subprogramele sunt incluse în programul principal și pot conține la rândul lor alte subprograme, rezultă că blocurile pot fi **imbricate** (incluse unul în altul). Această imbricare de blocuri este denumită **structura de bloc** a programului PASCAL.

Într-o structură, fiecărui bloc i se atașează câte un nivel de imbricare. Programul principal este considerat de nivel 0, un bloc definit în programul principal este de nivel 1. În general, un bloc definit în nivelul  $n$  este de nivelul  $n + 1$ .

Pentru exemplificare, în figura 1.2 este prezentată structura de bloc a programului P105.

### Pascal

```

Program P105;                                     {nivel 0}
  { Structura de bloc a programului }
var a : real;
{1}

procedure P(b : real);                         {nivel 1}
var c : real;
{2}

  procedure Q(d : integer);    {nivel 2}
  {3}
  var c : char;
  {4}
  begin
    c:=chr(d);
    writeln('In procedura Q  c=', c);
  end; {5}

begin
  writeln('b=', b);
  c:=b+1;
  writeln('In procedura P  c=', c);
  Q(35);
end; {6}

function F(x : real) : real;                  {nivel 1}
begin
  F:=x/2;
end;

begin
  a:=F(5);
  writeln('a=', a);
  P(a);
  readln;
end {7}.

```

Fig. 1.2. Structura de bloc a unui program PASCAL

De regulă, un bloc PASCAL include declarații de etichete, variabile, funcții, parametri și.a.m.d. O declarație introduce un nume, care poate fi o etichetă sau un identificator. O declarație dintr-un bloc poate redefini un nume declarat în exteriorul lui. În consecință, în diferite părți ale programului unul și același nume poate desemna obiecte diferite.

Prin **domeniul de vizibilitate** al unei declarații se înțelege textul de program, în care numele introdus desemnează obiectul specificat de declarația în studiu. Domeniul de vizibilitate începe imediat după terminarea declarației și se sfârșește odată cu textul blocului respectiv. Deoarece blocurile pot fi imbricate, domeniul de vizibilitate nu este neapărat o porțiune continuă din textul programului. Domeniul de vizibilitate al unei declarații dintr-un bloc inclus **acoperă** domeniul de vizibilitate al declarației ce implică același nume din blocul exterior.

De exemplu, în programul P105 domeniul de vizibilitate al declarației **var a : real** este textul cuprins între punctele marcate {1} și {7}. Domeniul de vizibilitate al declarației **var c : real** este format din două fragmente de text cuprinse între {2}, {3} și {5}, {6}. Domeniul de vizibilitate al declarației **var c : char** este textul cuprins între {4} și {5}.

Cunoașterea domeniilor de vizibilitate ale declarațiilor este necesară pentru determinarea obiectului curent desemnat de un nume.

De exemplu, identificatorul **c** din instrucțiunea

```
c := chr(d)
```

a programului P105 desemnează o variabilă de tip **char**. Același identificator din instrucțiunea

```
c := b+1
```

desemnează o variabilă de tip **real**.

De reținut că declarația unui nume de funcție/procedură se consideră terminată la sfârșitul antetului. Prin urmare, domeniul de vizibilitate al unei astfel de declarații include și corpul funcției/procedurii respective. Acest fapt face posibil **apelul recursiv**: în corpul funcției/procedurii aceasta poate fi referită, fiind vizibilă. Evident, declarația unui parametru formal este vizibilă numai în corpul subprogramului respectiv.

De exemplu, domeniul de vizibilitate al declarației **procedure Q** este textul cuprins între punctele marcate {3} și {6}. Domeniul de vizibilitate al declarației **d : integer** este textul cuprins între {3} și {5}.

## Întrebări și exerciții

- 1      Explică!** Cum se determină domeniul de vizibilitate al unei declarații?
- 2      Aplică!** Determinați domeniile de vizibilitate ale declarațiilor **b : real** și **x : real** din programul P105 (fig. 1.2).
- 3      Aplică!** Precizați structura de bloc a programului ce urmează. Indicați domeniul de vizibilitate al fiecărei declarații și determinați obiectele desemnate de fiecare apariție a identificatorilor **c** și **x**.

**P**ascal

```
Program P106;
  {Redefinirea constantelor }
const c=1;
function F1(x : integer) : integer;
begin
  F1:=x+c;
end; { F1 }

function F2(c : real) : real;
const x=2.0;
begin
  F2:=x+c;
end; { F2 }

function F3(x : char) : char;
const c=3;
begin
  F3:=chr(ord(x)+c);
end; { F3 }

begin
  writeln('F1=', F1(1));
  writeln('F2=', F2(1));
  writeln('F3=', F3('1'));
  readln;
end.
```

Ce va afișa la ecran programul în studiu?

**4 Exersează!** Determinați domeniile de vizibilitate ale identificatorilor **P** și **F** din programul P105 (fig. 1.2).

**5 Observă!** Comentați programul ce urmează:

**P**ascal

```
Program P107;
  { Eroare }
var a : real;

procedure P(x : real);
var a : integer;
begin
  a:=3.14;
  writeln(x+a);
end; { P }

begin
  a:=3.14;
  P(a);
end.
```

**6 Explică!** Cum se determină obiectul desemnat de apariția unui nume într-un program PASCAL?

## 1.5

## Domenii de vizibilitate (C/C++)

Corpul unui program sau subprogram se numește **bloc**. În unele limbaje de programare, de exemplu, în PASCAL, subprogramele sunt incluse în programul principal și pot conține la rândul lor alte subprograme, formând blocuri **imbricate** (incluse unul în altul).

În C/C++ corpul programului este compus din funcții. De obicei, funcțiile sunt descrise una după alta, finalizând cu descrierea funcției principale `main()`. Fiecare declarație din componenta unei funcții definește un obiect, de cele mai multe ori o variabilă, care este vizibilă și poate fi referită doar în interiorul acestei funcții. Mai mult ca atât, în funcțiile C/C++ pot fi definite blocuri care sunt părți componente ale unor instrucțiuni, de exemplu, ale ciclurilor **`for`** și **`while`**.

Definițiile din cadrul acestor blocuri devin efective doar în timpul execuției instrucțiunilor respective și sunt vizibile doar în interiorul acestora.

**Variabilele declarate în cadrul funcțiilor sau instrucțiunilor sunt numite *variabile locale*.**  
**Variabilele declarate în afara funcțiilor sunt numite *variabile globale*.**

Spre deosebire de variabilele locale, care sunt vizibile și pot fi referite doar în cadrul funcțiilor sau instrucțiunilor în care sunt definite, variabilele globale sunt vizibile și pot fi referite în oricare dintre funcțiile ce urmează după declarația variabilelor respective.

### Exemplul 1:

Se consideră următorul program:

C++

```
// Program P102
// Domenii de vizibilitate
#include <iostream>
#include <math.h>
using namespace std;

void Tipartablou(int x[], int n)
{
    for (int i = 0; i < n; i++) cout << x[i] << " ";
    cout << endl;
    return;
}

bool Esteprim(int q)
{
    int h = 0;
    for (int i = 1; i <= q; i++)
        if (q % i == 0) h++;

    if (h == 2) return true; else return false;
}
```

```

void Numereprime(int x[], int n)
{
    int p[n];
    int j = 0;
    for (int i = 0; i < n; i++)
        if (Esteprim(x[i]) == true)
    {
        p[j] = x[i]; j++;
    }
    Tipartablou(p, j);
    return;
}

int main()
{
    int a[10] = {27, 12, 13, 8, 9, 148, 7, 23, 55, 5};
    int n = 10;
    Tipartablou(a, n);
    Numereprime(a, n);
    return 0;
}

```

În acest program sunt declarate următoarele funcții:

**Tipartablou** – funcție fără tip, care afișează la ecran primele n elemente ale tabloului x. În această funcție este declarată variabila locală i, care poate fi utilizată doar în interiorul ciclului **for**.

**Esteprim** – funcție de tip boolean, care returnează valoarea **true** dacă numărul q este un număr prim și **false** în caz contrar. În această funcție este declarată variabila locală i, care poate fi utilizată doar în interiorul ciclului **for**, și variabila h, care poate fi utilizată în orice instrucțiune a funcției respective.

**Numereprime** – funcție fără tip, care afișează la ecran doar numerele prime din tabloul x. În funcție este declarată variabila locală i, care poate fi utilizată doar în interiorul ciclului **for**, variabila j, care poate fi utilizată în orice instrucțiune a funcției respective, la fel ca și tabloul p[n], dimensiunea căruia este determinată de valoarea parametrului, transmis în funcție.

**main** – funcția principală, care atribuie valori elementelor tabloului a, afișează toate aceste valori (apelând funcția **Tipartablou**) și, ulterior, afișează doar numerele prime din tabloul a (apelând funcția **Numereprime**). În funcția **main** sunt definite două variabile: tabloul a și variabila simplă n de tip **int**.

Accentuăm faptul că din cadrul unuia și același program, dar în funcții diferite, programatorul poate declara variabile ce au aceeași denumire. Variabila concretă ce va fi utilizată în calcule depinde de locul în care apare referirea respectivă.

De exemplu, unul și același nume de variabilă i apare în declarațiile din trei funcții: **Tipartablou**, **Esteprim** și **Numereprime**. Deși au aceeași denumire i, variabilele respective sunt locale în blocurile în care au fost declarate și, evident, valorile lor în niciun fel nu depind una de alta.

## Întrebări și exerciții

- 1 **Explică!** Cum se determină domeniul de vizibilitate al unei declarații?
- 2 **Aplică!** Determinați domeniile de vizibilitate ale declarațiilor **int j = 0;** și **int i = 0;** din funcția Numereprime din programul P102\_C/C++.
- 3 **Aplică!** Precizați structura programului ce urmează. Indicați domeniul de vizibilitate al fiecărei declarații și determinați obiectele desemnate de fiecare apariție a identificatorilor **c** și **x.**

### C++

```
// Program P103
// Redefinirea constantelor
#include <iostream>
#define C 1
using namespace std;

int F1(int x)
{
    return x + C;
}

double F2(double c)
{
    #define X 2.0
    return X + c;
}

char F3(char x)
{
    #define C '3'
    return (char)(int(x) + C);
}

int main()
{
    cout << "F1=" << F1(1) << endl;
    cout << "F2=" << F2(1) << endl;
    cout << "F3=" << F3('1') << endl;
}
```

Ce va afișa la ecran programul în studiu?

- 4 **Exersează!** Determinați domeniile de vizibilitate ale identificatorilor **p**, **a** și **n** din programul P102.
- 5 **Observă!** Comentați programul ce urmează:

### C++

```
// Program P104
// Eroare
#include <iostream>
using namespace std;
```

```

double a;

void P(double x)
{
    int a;
    a = 3.14;
    cout << x + a;
}

int main()
{
    a = 3.14;
    P(a);
}

```

6

**Explică!**

Cum se determină obiectul desemnat de apariția unui nume într-un program C/C++?

1.6

## Comunicarea prin variabile globale

### Pascal

Execuția unui apel de subprogram presupune transmiterea datelor de prelucrat funcției sau procedurii respective. După executarea ultimei instrucțiuni din subprogram, rezultatele produse trebuie întoarse în locul de apel. Cunoaștem deja că datele de prelucrat și rezultatele produse pot fi transmise prin parametri. Parametrii formali se specifică în antetul funcției sau procedurii, iar parametrii actuali – în locul apelului.

În completare la modul de transmitere a datelor prin parametri, limbajul PASCAL permite comunicarea prin variabile globale.

Orice variabilă este locală în subprogramul în care a fost declarată. O variabilă este **globală relativ la un subprogram** atunci când ea este declarată în programul sau subprogramul ce îl cuprinde fără să fie redeclarată în subprogramul în studiu. Întrucât variabilele globale sunt cunoscute atât în subprogram, cât și în afara lui, ele pot fi folosite pentru transmiterea datelor de prelucrat și returnarea rezultatelor.

#### *Exemplu:*

### Pascal

```

Program P108;
  {Comunicarea prin variabile globale }
  var a,           {variabila globală în P }
    b : real;       {variabila globală în P, F }

  procedure P;
  var c : integer; {variabila locală în P }
  begin
    c:=2;
    b:=a*c;
  end; { P }

```

```

function F : real;
var a : 1..5;      {variabila locală în F }
begin
    a:=3;
    F:=a+b;
end; { F }

begin
    a:=1;
    P;
    writeln(b);      {se afiseaza 2.0000000000E+00 }
    writeln(F);      {se afiseaza 5.0000000000E+00 }
    readln;
end.

```

### C++

```

// Program P108
// Comunicarea prin variabile globale
#include <iostream>
using namespace std;
double a, b; // variabile globale
void P()
{
    int c = 2; // variabila locală în P
    b = a * c;
    return;
}
double F()
{
    int a = 3; // variabila locală în F
    return a + b;
}
int main()
{
    a = 1;
    P();
    cout << b << endl; // se afiseaza 2
    cout << F() << endl; // se afiseaza 5
    return 0;
}

```

Datele de prelucrat se transmit procedurii P prin variabila globală a. Rezultatul produs de procedură se returnează în blocul de apel prin variabila globală b. Valoarea argumentului funcției F se transmite prin variabila globală b. Menționăm că variabila a este locală în F și nu poate fi folosită pentru transmiterea datelor în această funcție.

De obicei, comunicarea prin variabile globale se utilizează în cazurile în care mai multe subprograme prelucrează aceleași date. Pentru exemplificare, amintim funcțiile cu argumente de tip *tablou*, procedurile care prelucrează tablouri și fișiere de angajați, persoane, elevi etc.

## Întrebări și exerciții

- 1 Explicați termenii variabilă globală relativ la un subprogram și variabilă locală într-un subprogram.
- 2 **Aplică!** (PASCAL) Numiți variabilele globale și variabilele locale din programul P105 (fig. 1.2).
- 3 **Cercetează!** Poate fi oare o variabilă locală și, în același timp, o variabilă globală relativ la un subprogram?
- 4 **Aplică!** Numiți variabilele globale și variabilele locale din programul ce urmează. Ce va afișa la ecran acest program?

### Pascal

```
Program P109;
  { Comunicarea prin variabile globale }
var a : integer;

procedure P;
var b, c, d : integer;

procedure Q;
begin
  c:=b+1;
end; { Q }

procedure R;
begin
  d:=c+1;
end; { R }
begin
  b:=a;
  Q;
  R;
  a:=d;
end; { P }

begin
  a:=1;
  P;
  writeln(a);
  readln;
end.
```

### C++

```
// Program P109
// Comunicarea prin variabile globale
#include <iostream>
using namespace std;

int a, b;

void R()
```

```

    {
        a++;
        cout << " a in R() = "<< a << endl;
        return;
    }

void Q()
{
    b = a + 2;
    cout << " a in Q() = "<< a << endl;
    cout << " b in Q() = "<< b << endl;
    return;
}

void P()
{
    int b, d = 6;
    b = a;
    Q();
    R();
    a = d;
    cout << " b in P() = "<< b << endl;
    cout << " a in P() = "<< a << endl;
    return;
}

int main()
{
    a = 1;
    P();
    cout << " a in main() = "<< a << endl;
    cout << " b in main() = "<< b << endl;
    return 0;
}

```

5

**Programează!**

Se consideră declarațiile:

**Pascal**

```

Type Ora=0..23;
      Grade=-40..+40;
      Temperatura=array [Ora] of Grade;

```

**C++**

```

#define Ora 24
int Grade;
int Temperatura[Ora];

```

Componentele unei variabile de tip Temperatura reprezintă temperaturile măsurate din oră în oră pe parcursul a 24 de ore. Elaborați o procedură care:

- indică maximumul și minimumul temperaturii;
- indică ora (orele) la care s-a înregistrat o temperatură maximă;
- înscrive într-un fișier text ora (orele) la care s-a înregistrat o temperatură minimă.

Comunicarea cu procedurile respective se va face prin variabile globale.

6

**Integrează cunoștințele și aplică!**

Se consideră fișiere arbitrare de tip text. Elaborați o funcție care:

- returnează numărul de linii dintr-un fișier;
- calculează numărul de vocale dintr-un text;
- calculează numărul de cuvinte dintr-un text (cuvintele reprezintă șiruri de caractere separate prin spațiu sau sfârșit de linie);
- returnează lungimea medie a liniilor din text;
- calculează lungimea medie a cuvintelor din text;
- returnează numărul semnelor de punctuație din text.

Comunicarea cu funcțiile respective se va face prin variabile globale.

1.7

## Efecte colaterale

Destinația unei funcții este să întoarcă ca rezultat o singură valoare. În mod obișnuit, argumentele se transmit funcției prin parametri-valoare, iar rezultatul calculat se returnează în locul de apel prin numele funcției. În completare, limbajul PASCAL permite transmiterea argumentelor prin variabile globale și parametri-variabilă.

Prin **efect colateral** se înțelege o atribuire (în corpul funcției) a unei valori la o variabilă globală sau la un parametru formal variabilă. Efectele colaterale pot influența în mod neașteptat execuția unui program și complică procesele de depanare.

Prezentăm în continuare exemple defectuoase de programare care folosesc funcții cu efecte colaterale.

### Pascal

```
Program P110;
{Efect colateral - atribuire la o variabila globala}
var a : integer; { variabila globala }

function F(x : integer) : integer;
begin
  F:=a*x;
  a:=a+1;      {atribuire defectuoasa }
end; { F }
begin
  a:=1;
  writeln(F(1)); { se afiseaza 1 }
  writeln(F(1)); { se afiseaza 2 }
  writeln(F(1)); { se afiseaza 3 }
  readln;
end.
```

## C++

```
// Program P110
// Efect colateral - atribuire la o variabila globala

#include <iostream>
using namespace std;

int a; // variabila globala

int F(int x)
{
    int tmp = a * x;
    a++; // atribuire defectuoasa
    return tmp;
}

int main()
{
    a = 1;
    cout << F(1) << endl; // se afiseaza 1
    cout << F(1) << endl; // se afiseaza 2
    cout << F(1) << endl; // se afiseaza 3
}
```

În programul P110 funcția F returnează valoarea expresiei  $a \cdot x$ . Pe lângă asta însă, atribuirea  $a := a + 1$  altereză valoarea variabilei globale a. În consecință, pentru una și aceeași valoare 1 a argumentului x funcția returnează rezultate diferite, fapt ce nu se încadrează în conceptul ușual de funcție.

## Pascal

```
Program P111;
{Efect colateral - atribuire la un parametru formal}
var a : integer;

function F(var x : integer) : integer;
begin
    F:=2*x;
    x:=x+1; { atribuire defectuoasa }
end; { F }

begin
    a:=2;
    writeln(F(a)); { se afiseaza 4 }
    writeln(F(a)); { se afiseaza 6 }
    writeln(F(a)); { se afiseaza 8 }
    readln;
end.
```

**C++**

```
// Program P111
// Efect colateral - atribuire la un parametru formal

#include <iostream>
using namespace std;
int a; // variabila globala

int F(int* x)
{
    *x = 2 * (*x);
    return *x++;
}

int main()
{
    a = 2;
    F(&a);
    cout << a << endl; // se afiseaza 4
    F(&a);
    cout << a << endl; // se afiseaza 8
    F(&a);
    cout << a << endl; // se afiseaza 16
}
```

În programul P111 funcția F returnează valoarea expresiei  $2*x$ . Întrucât  $x$  este un parametru formal variabilă, atribuirea  $x := x + 1$  schimbă valoarea parametrului actual din apel, și anume a variabilei  $a$  din programul principal. Faptul că apelurile textual identice  $F(a)$ ,  $F(a)$  și  $F(a)$  returnează rezultate ce diferă poate crea confuzii în procesul depanării.

În cazul procedurilor, atribuirile asupra variabilelor globale produc efecte colaterale similare celor discutate pentru astfel de atribuiră la funcții. Întrucât mijlocul-standard de întoarcere de rezultate din procedură este prin parametri formali variabilă, atribuirile asupra unor astfel de parametri nu sunt considerate ca efecte colaterale.

Efectele colaterale introduc abateri de la procesul-standard de comunicare, prin care variabilele participante sunt desemnate explicit ca parametri formali în declarație și parametri actuali în apel. Consecințele efectelor colaterale se pot propaga în domeniul de vizibilitate al declarărilor globale și pot interfecla cu cele similare produse la execuția altor proceduri și funcții. În astfel de condiții, utilizarea variabilelor globale devine riscantă. Prin urmare, la elaborarea programelor complexe se vor aplica următoarele recomandări:

1. Comunicarea funcțiilor cu mediul de chemare se va face prin transmiterea de date spre funcție prin parametri formali valoare și întoarcerea unui singur rezultat prin numele ei.
2. Comunicarea procedurilor cu mediul de chemare se va face prin transmiterea de date prin parametri formali valoare sau variabilă și întoarcerea rezultatelor prin parametri formali variabilă.
3. Variabilele globale pot fi folosite pentru transmiterea datelor în subprograme, însă valorile lor nu trebuie să fie modificate de acestea.

## Întrebări și exerciții

1 **Explică!** Care este cauza efectelor colaterale? Ce consecințe pot avea aceste efecte?

2 **Cercetează!** Precizați ce vor afișa la ecran programele ce urmează:

### Pascal

```
Program P112;
{Efecte colaterale }
var a, b : integer;

function F(x : integer) : integer;
begin
  F:=a*x;
  b:=b+1;
end; { F }

function G(x : integer) : integer;
begin
  G:=b+x;
  a:=a+1;
end; { G }

begin
  a:=1; b:=1;
  writeln(F(1));
  writeln(G(1));
  writeln(F(1));
  writeln(G(1));
  readln;
end.
```

### C++

```
// Program P112
// Efecte colaterale
#include <iostream>
using namespace std;

int a, b;

int F(int x)
{
  b++;
  return a + x;
}
```

```
int G(int x)
{
    a++;
    return b + x;
}

int main()
{
    a = 1; b = 1;
    cout << F(1) << endl;
    cout << G(1) << endl;
    cout << F(1) << endl;
    cout << G(1) << endl;
}
```

## Pascal

```
Program P113;
{Efekte colaterale }

var a : integer;
    b : real;

function F(var x : integer) : integer;
begin
    F:=x;
    x:=x+1;
end; { F }

procedure P(x,y:integer; var z:real);
begin
    z:=x/y;
end; { P }

begin
    a:=1;
    P(F(a), a, b);
    writeln(a, ' ', b);
    readln;
end.
```

## C++

```
// Program P113
// Efecte colaterale

#include <iostream>
using namespace std;

int a;
double b;

int F(int* x)
{
    *x = *x + 2;
    return *x;
}

void P(int x, double y, double* z)
{
    *z = x / y;
}

int main()
{
    a = 1;
    P(F(&a), a / 2.0, &b);
    cout << a << " " << b << endl;
}
```

## Pascal

```
Program P114;
{Efecte colaterale }

var a, b : real;

procedure P(var x, y : real);
{Interschimbarea valorilor variabilelor x, y }
begin
    a:=x;
    x:=y;
    y:=a;
end; { P }
```

```
begin
  a:=1; b:=2;
  P(a, b);
  writeln(a, b);
  a:=3; b:=4;
  P(a, b);
  writeln(a, b);
  readln;
end.
```

## C++

```
// Program P114
// Efecte colaterale

#include <iostream>
using namespace std;

double a, b;

void P(double* x, double* y)
    // Interschimbarea valorilor variabilelor x, y
{
    a = *x;
    *x = *y;
    *y = a;
}

int main()
{
    a = 1; b = 2;
    P(&a, &b);
    cout << a << " " << b << endl;
    a = 3; b = 4;
    P(&a, &b);
    cout << a << " " << b << endl;
}
```

3

**Explică!**

Cum pot fi evitate efectele colaterale?

## Recursia

**Recursia** se definește ca o situație în care un subprogram se autoapeleză fie direct, fie prin intermediul altrei funcții sau proceduri. Subprogramul care se autoapeleză se numește **recursiv**.

De exemplu, funcția *factorial*

$$f(n) = \begin{cases} 1, & \text{dacă } n = 0; \\ n \cdot f(n-1), & \text{dacă } n > 0 \end{cases}$$

poate fi exprimată în PASCAL, urmând direct definiția, în forma

### Pascal

```
function F(n : integer) : integer;
begin
  if n=0 then F:=1
  else F:=n*F(n-1)
end; {F}
```

### C++

```
int F(int n)
{
  if (n == 0) return 1;
  else return n * F(n - 1);
}
```

Efectul unui apel  $F(7)$  este declanșarea unui lanț de apeluri ale funcției  $F$  pentru parametrii actuali  $6, 5, \dots, 2, 1, 0$ :

$F(7) \rightarrow F(6) \rightarrow F(5) \rightarrow \dots \rightarrow F(1) \rightarrow F(0).$

Apelul  $F(0)$  determină evaluarea directă a funcției și oprirea procesului repetitiv; urmează revenirile din apeluri și evaluarea lui  $F$  pentru  $1, 2, \dots, 6, 7$ , ultima valoare fiind întoarcerea în locul primului apel.

Funcția

$$fib(n) = \begin{cases} 0, & \text{dacă } n = 0; \\ 1, & \text{dacă } n = 1; \\ fib(n-1) + fib(n-2), & \text{dacă } n > 1 \end{cases}$$

are ca valori numerele lui *Fibonacci*. Urmând definiția, obținem:

### Pascal

```
function Fib(n:integer):integer;
begin
  if n=0 then Fib:=0
  else if n=1 then Fib:=1
    else Fib:=Fib(n-1)+Fib(n-2)
end; {Fib}
```

**C++**

```
int Fib(int n)
{
    if (n==0) return 0;
    else if (n == 1) return 1;
    else return Fib(n - 1) + Fib(n - 2);
}
```

Fiecare apel al funcției Fib pentru  $n > 1$  generează două apeluri  $\text{Fib}(n-1)$ ,  $\text{Fib}(n-2)$  și.a.m.d., de exemplu:

$\text{Fib}(4) \rightarrow$

$\text{Fib}(3), \text{Fib}(2) \rightarrow$

$\text{Fib}(2), \text{Fib}(1), \text{Fib}(1), \text{Fib}(0) \rightarrow$

$\text{Fib}(1), \text{Fib}(0).$

Din exemplele în studiu se observă că recursia este utilă pentru programarea unor calcule repetitive. Repetiția este asigurată prin execuția unui subprogram care conține un apel la el însuși: când execuția ajunge la acest apel, este declanșată o nouă execuție și.a.m.d.

Evident, orice subprogram recursiv trebuie să includă condiții de oprire a procesului repetitiv. De exemplu, în cazul funcției factorial  $F$  procesul repetitiv se oprește când  $n$  ia valoarea 0; în cazul funcției Fibonacci  $\text{Fib}$  procesul se oprește când  $n$  ia valoarea 0 sau 1.

La orice apel de subprogram în memoria calculatorului vor fi depuse următoarele informații:

- valorile curente ale parametrilor transmiși prin valoare;
- locațiile (adresele) parametrilor-variabilă;
- adresa de return, adică adresa instrucțiunii ce urmează după apel.

Prin urmare, la apeluri recursive spațiul ocupat din memorie va crește rapid, riscând depășirea capacitații de memorare a calculatorului. Astfel de cazuri pot fi evitate, înlocuind recursia prin iterare (instrucțiunile **for**, **while**, **repeat**). Pentru exemplificare, prezentăm o formă nerecursivă a funcției *factorial*:

**Pascal**

```
function F(n: Natural): Natural;
var i, p : Natural;
begin
    p:=1;
    for i:=1 to n do p:=p*i;
    F:=p;
end; {F}
```

**C++**

```
int F(int n)
{
    int i, p = 1;
    for (i = 1; i <= n; i++)
        p = p * i;
    return p;
}
```

Recursia este deosebit de utilă în cazurile în care elaborarea algoritmilor nerecursivi este foarte complicată: translatarea programelor PASCAL în limbajul cod-mașină, grafica pe calculator, recunoașterea imaginilor și.a.

## Întrebări și exerciții

**1 Explică!** Cum se execută un subprogram recursiv? Ce informații se depun în memoria calculatorului la execuția unui apel recursiv?

**2 Observă!** Care este diferența dintre recursie și iterație?

**3 Integreză cunoștințele și aplică!** Elaborați o formă nerecursivă a funcției lui Fibonacci.

**4 Programează!** Scrieți un subprogram recursiv care:

- a) calculează suma  $S(n) = 1 + 3 + 5 + \dots + (2n - 1)$ ;
- b) calculează produsul  $P(n) = 1 \times 4 \times 7 \times \dots \times (3n - 2)$ ;
- c) inversează un sir de caractere;
- d) calculează produsul  $P(n) = 2 \times 4 \times 6 \times \dots \times 2n$ .

**5 Experimentează!** Elaborați un program care citește de la tastatură numerele naturale  $m, n$  și afișează la ecran valoarea funcției lui Ackermann:

$$a(m, n) = \begin{cases} n+1, & \text{dacă } m = 0; \\ a(m-1, 1), & \text{dacă } n = 0; \\ a(m-1, a(m, n-1)), & \text{dacă } m > 0 \text{ și } n > 0. \end{cases}$$

Calculați  $a(0, 0)$ ,  $a(1, 2)$ ,  $a(2, 1)$  și  $a(2, 2)$ . Încercați să calculați  $a(4, 4)$  și  $a(10, 10)$ . Explicați mesajele afișate la ecran.

**6 Programează!** Se consideră declarația:

**Pascal**

```
type Vector=array [1..20] of integer;
```

**C++**

```
int Vector[20];
```

Elaborați un subprogram recursiv care:

- a) afișează componentele vectorului la ecran;
- b) calculează suma componentelor;
- c) inversează componentele vectorului;
- d) calculează suma componentelor pozitive;
- e) verifică dacă cel puțin o componentă a vectorului este negativă;
- f) calculează produsul componentelor negative;
- g) verifică dacă cel puțin o componentă a vectorului este egală cu un număr dat.

7

**Creează!**

Elaborați o formă nerecursivă a funcției ce urmează:

**Pascal**

```
function S(n:Natural):Natural;
begin
  if n=0 then S:=0
  else S:=n+S(n-1)
end; {S}
```

**C++**

```
int S(int n)
{
  if (n == 0) return 0;
  else return n + S(n - 1);
}
```

8

**Integrează cunoștințele și aplică!**

Scrieți o funcție recursivă care returnează valoarea true dacă sirul de caractere s este conform definiției:

$\langle \text{Număr} \rangle ::= \langle \text{Cifră} \rangle \mid \langle \text{Cifră} \rangle \langle \text{Număr} \rangle$

*Indicație.* Forma unei astfel de funcții derivă din formula metalingvistică. Varianta nerecursivă

**Pascal**

```
function N(s : string) : boolean;
var i : integer;
    p : boolean;
begin
  p:=(s<> '');
  for i:=1 to length(s) do
    p:=p and (s[i] in ['0'...'9']);
  N:=p;
end;
```

**C++**

```
bool N(char* s[])
{
  for (int i = 0; i < s.length(); i++)
    if (s[i] < '0' || s[i] > '9') return false;
  return true;
}
```

derivă din definiția

$\langle \text{Număr} \rangle ::= \langle \text{Cifră} \rangle \{ \langle \text{Cifră} \rangle \}$

9

**Integrează cunoștințele și aplică!**

Se consideră următoarele formule metalingvistice:

$\langle \text{Număr} \rangle ::= \langle \text{Cifră} \rangle \{ \langle \text{Cifră} \rangle \}$

$\langle \text{Semn} \rangle ::= + \mid -$

$\langle \text{Expresie} \rangle ::= \langle \text{Număr} \rangle \mid \langle \text{Expresie} \rangle \langle \text{Semn} \rangle \langle \text{Expresie} \rangle$

Scrieți o funcție recursivă care returnează valoarea true dacă sirul de caractere s este conform definiției unității lexicale  $\langle \text{Expresie} \rangle$ .

## 1.9

## Sintaxa declarațiilor și apelurilor de subprograme

### Pascal

În general, definirea unei funcții PASCAL se face cu ajutorul următoarelor formule metalingvistice:

```

<Funcție> ::= <Antet funcție>; <Corp>
            | <Antet funcție>; <Directivă>
            | function <Identificator>; <Corp>
<Antet funcție> ::= function <Identificator> [<Listă parametri formalii>] : <Identificator>
  
```

Diagramele sintactice sunt prezentate în figura 1.3.

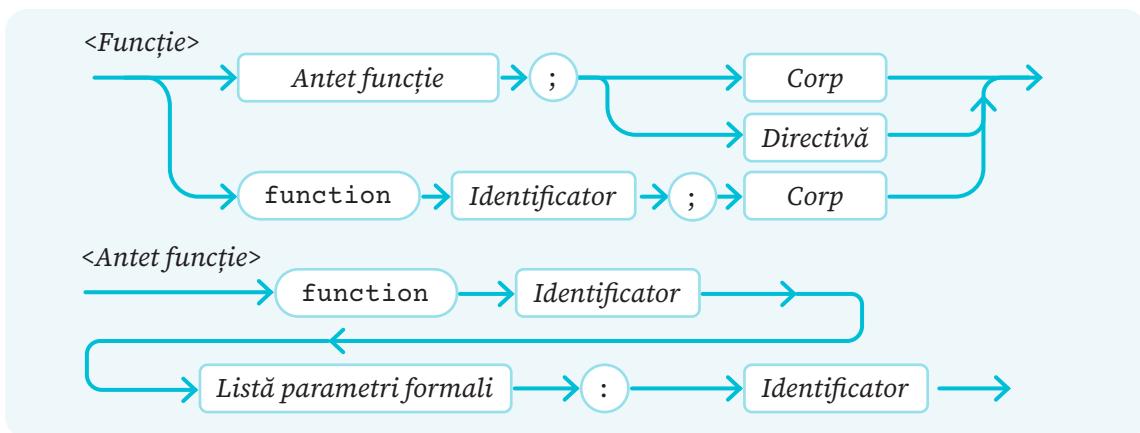


Fig. 1.3. Sintaxa declarațiilor de funcții

Procedurile PASCAL se definesc cu ajutorul următoarelor formule:

```

<Procedură> ::= <Antet procedură>; <Corp> | <Antet procedură>; <Directivă>
                  | procedure <Identificator>; <Corp>
  
```

<Antet procedură> := **procedure** <Identificator> [<Listă parametri formalii>]

Diagramele sintactice sunt prezentate în figura 1.4.

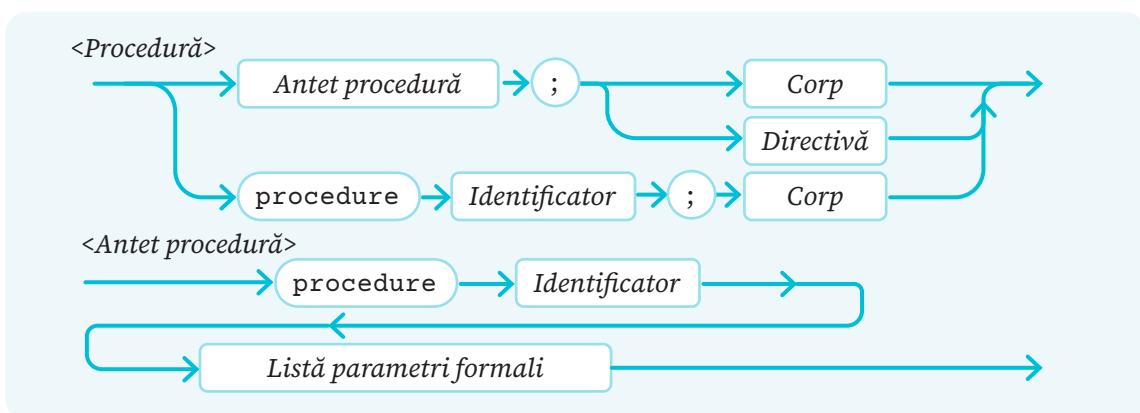


Fig. 1.4. Sintaxa declarațiilor de proceduri

**Listele de parametri formali** au următoarea sintaxă:

*<Listă parametri formali>* ::=

(*<Parametru formal>* { ; *<Parametru formal>* })

*<Parametru formal>* ::=

[**var**] *<Identificator>* { , *<Identificator>* } : *<Identificator>*

| *<Antet funcție>*

| *<Antet procedură>*

Diagrama sintactică este prezentată în figura 1.5.

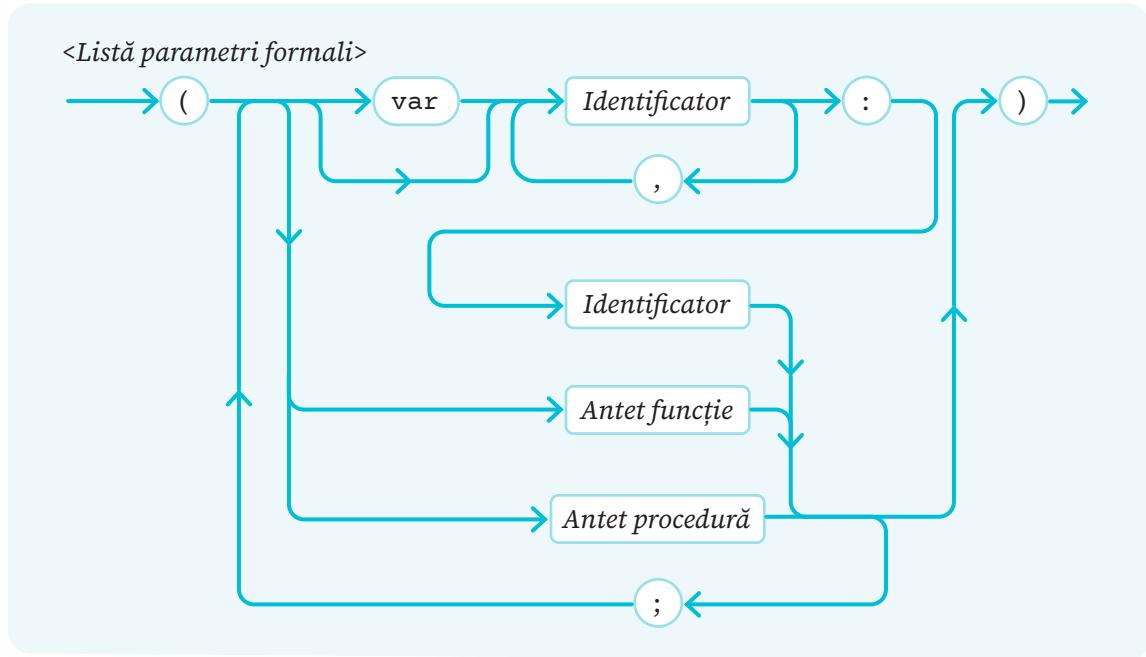


Fig. 1.5. Diagrama sintactică *<Listă parametri formali>*

Amintim că în lipsa cuvântului-cheie **var** identificatorii din listă specifică **parametrii-valoare**. Cuvântul **var** prefixează **parametrii-variabilă**. Antetul unei funcții (proceduri) din listă specifică un **parametru-funcție (procedură)**. În Turbo PASCAL astfel de parametri se declară explicit ca aparținând unui **tip procedural** și au forma parametrilor-valoare. Limbajul PASCAL extinde sensul ușual al noțiunii de funcție, permitând returnarea valorilor nu numai prin numele funcției, ci și prin parametri-variabilă.

Un **apel de funcție** are forma:

*<Apel funcție>* ::= *<Nume funcție>* [*<Listă parametri actuali>*]

iar o instrucțiune **apel de procedură**:

*<Apel procedură>* ::= *<Nume procedură>* [*<Listă parametri actuali>*]

**Parametrii actuali** se specifică cu ajutorul formulelor:

*<Listă parametri actuali>* ::= (*<Parametru actual>* { , *<Parametru actual>* })

*<Parametru actual>* ::= *<Expresie>* | *<Variabilă>* | *<Nume funcție>* | *<Nume procedură>*

Diagrama sintactică este prezentată în figura 1.6.

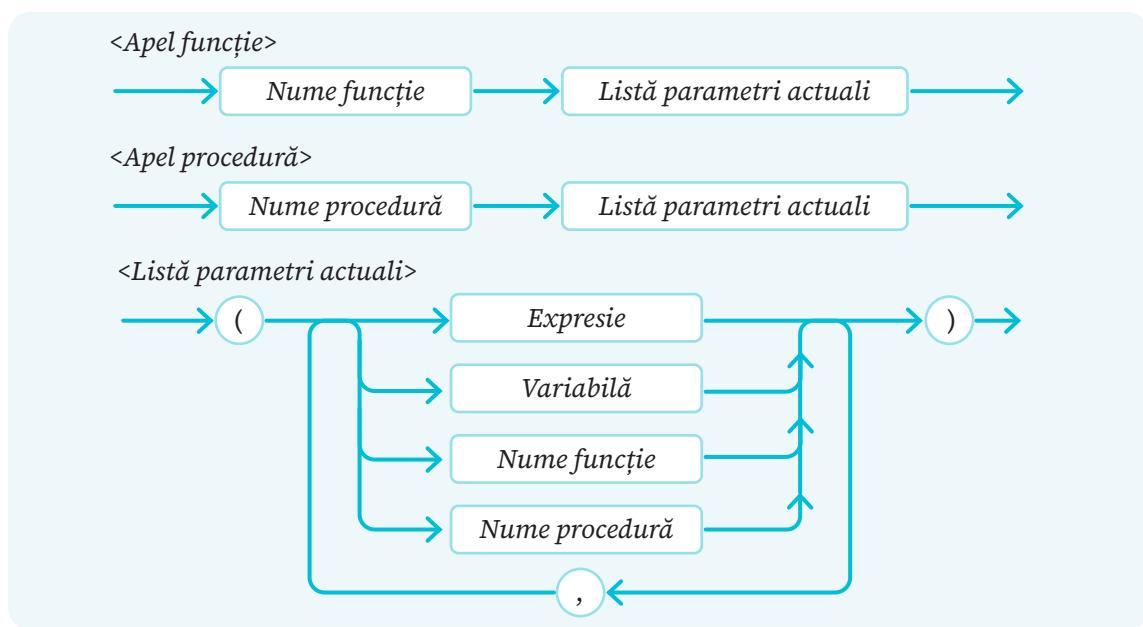


Fig. 1.6. Sintaxa apelurilor de funcții și proceduri

Corespondența dintre un parametru actual și parametrul formal se face prin poziția ocupată de acesta în cele două liste.

În cazul unui **parametru-valoare**, drept parametru actual poate fi utilizată orice expresie, în particular, o constantă sau o variabilă. Expressia respectivă trebuie să fie compatibilă din punctul de vedere al atribuirii cu tipul parametrului formal. Modificările parametrilor-valoare nu se transmit în exteriorul subprogramului.

În cazul unui **parametru-variabilă**, drept parametri actuali pot fi utilizate numai variabile. Modificările parametrilor-variabilă se transmit în exteriorul subprogramului.

În cazul unui **parametru-funcție (procedură)**, drept parametru actual poate fi utilizat orice nume de funcție (procedură) antetul căreia are forma specificată în lista parametrilor formali.

## C++

În general, definirea unei funcții C/C++ se face cu ajutorul următoarelor formule metalingvistice:

```

<Funcție> ::= <Antet funcție> <Corp>
<Antet funcție> ::= <Tip> <Identificator> [<Listă parametri formali>]
<Tip> ::= void | int | long | float | double | long double | char | *
    
```

*Notă.* Multitudinea tipurilor de date utilizate în limbajele C/C++ nu permite descrierea completă a acestora în cadrul unei formule metalingvistice suficient de compacte. În scopuri didactice, în formula metalingvistică de mai sus sunt incluse doar tipurile frecvent utilizate.

**Listele de parametri formali** au următoarea sintaxă:

```

<Listă parametri formali> ::= <Parametru formal> {, <Parametru formal>}
<Parametru formal> ::= 
    <Tip> [* | &] <Identificator> | <Antet funcție>
    {, <Tip> [* | &] <Identificator>} | <Antet funcție>
    
```

Un **apel de funcție**, inclusiv instrucțiunea **apel de funcție** are forma:  
 $\langle \text{Apel funcție} \rangle ::= \langle \text{Nume funcție} \rangle ([\langle \text{Listă parametri actuali} \rangle])$

**Parametrii actuali** se specifică cu ajutorul formulelor:

$\langle \text{Listă parametri actuali} \rangle ::= \langle \text{Parametru actual} \rangle \{, \langle \text{Parametru actual} \rangle\}$   
 $\langle \text{Parametru actual} \rangle ::= \langle \text{Expresie} \rangle | \langle \text{Variabilă} \rangle | \langle \text{Apel funcție} \rangle$

Corespondența dintre un parametru actual și parametrul formal se face prin poziția ocupată de aceștia în cele două liste.

**Atenție!** Spre deosebire de limbajul PASCAL, care a fost conceput pentru a învăța în școală bazele algoritmizării și programării, limbajele C/C++ au fost concepute pentru programarea profesională. În consecință, sintaxa și semantica limbajelor C/C++ sunt cu mult mai complexe decât ale limbajului PASCAL. Acest fapt face imposibilă includerea într-un manual școlar a tuturor formulelor metalingvistice și a diagramele sintactice respective. Pentru a se familiariza cu aceste formule și diagrame, recomandăm acelor elevi, care în viitor intenționează să devină programatori profesioniști, să consulte sistemele de asistență ale mediilor de programare C/C++.

## Întrebări și exerciții

- 1 **Explică!** (PASCAL) Când se utilizează declarațiile de forma  
**function**  $\langle \text{Identificator} \rangle ; \langle \text{Corp} \rangle$  ?
- 2 **Explică!** (C/C++) Se consideră formula metalingvistică ce descrie antetul unei funcții C/C++. Ce tip de funcție specifică descrierea  
**void**  $\langle \text{Identificator} \rangle ([\langle \text{Listă parametri formalii} \rangle])$ ?
- 3 **Explică!** (C/C++) Se consideră următoarele formule metalingvistice ce definesc apeluri de funcții:
  - a)  $\langle \text{Antet funcție} \rangle ::= \langle \text{Tip} \rangle \langle \text{Identificator} \rangle [\langle \text{Listă parametri formalii} \rangle]$   
 $\langle \text{Listă parametri formalii} \rangle ::= (\langle \text{Parametru formal} \rangle \{, \langle \text{Parametru formal} \rangle\})$
  - b)  $\langle \text{Antet funcție} \rangle ::= \langle \text{Tip} \rangle \langle \text{Identificator} \rangle ([\langle \text{Listă parametri formalii} \rangle])$   
 $\langle \text{Listă parametri formalii} \rangle ::= \langle \text{Parametru formal} \rangle \{, \langle \text{Parametru formal} \rangle\}$
 Se observă că în ambele formule metalingvistice lista de parametri formalii este optională. Care este deosebirea dintre apelurile respective în cazurile în care parametrii formalii lipsesc?
- 4 **Explorează!** (PASCAL) Indicați pe diagramele sintactice din *figurile 1.3 și 1.5* drumurile care corespund declarațiilor de funcții din programul P106, paragraful 1.4.
- 5 **Explică!** Care este diferența dintre un parametru-valoare și un parametru-variabilă?
- 6 **Explorează!** (PASCAL) Indicați pe diagramele sintactice din *figurile 1.4 și 1.5* drumurile care corespund declarațiilor de proceduri din programul P101, paragraful 1.3.
- 7 **Explorează!** (PASCAL) Indicați pe diagramele sintactice din *figurile 1.3–1.6* drumurile care corespund declarațiilor și apelurilor de subprograme din programul P105, paragraful 1.4.



# Tehnici de programare

2.1

## Complexitatea algoritmilor

Valoarea practică a programelor de calculator depinde în mod decisiv de complexitatea algoritmilor ce stau la baza lor. Amintim că algoritmul reprezintă o succesiune finită de operații (instructiuni, comenzi) cunoscute, care, fiind executate într-o ordine bine stabilită, furnizează soluția unei probleme.

E cunoscut faptul că unul și același algoritm poate fi descris prin diverse metode: scheme logice, formule matematice, texte scrise într-un limbaj de comunicare între oameni, cu ajutorul limbajelor de programare. Evident, și în acest manual algoritmii pe care îi vom studia vor fi descriși cu ajutorul mijloacelor oferite de limbajele PASCAL și C/C++ : instructiuni, funcții, proceduri și programe ce pot fi derulate pe calculator.

**Complexitatea algoritmului** se caracterizează prin *necesarul de memorie și durata de execuție*. Metodele de estimare a acestor indicatori se studiază într-un compartiment special al informaticii, denumit **analiza algoritmilor**. În cadrul acestui compartiment se utilizează următoarele notații:

$n$  – un număr natural ce caracterizează mărimea datelor de intrare ale unui algoritm. În majoritatea cazurilor  $n$  reprezintă numărul de elemente ale unei mulțimi, gradul unei ecuații, numărul de componente ale unui tablou și.a.;

$V(n)$  – volumul de memorie internă necesară pentru păstrarea datelor cu care operează algoritmul;

$T(n)$  – timpul necesar executării algoritmului. De obicei, în acest timp nu se include durata operațiilor de introducere a datelor inițiale și de extragere a rezultatelor.

Evident, volumul de memorie  $V(n)$  și timpul de execuție  $T(n)$  depind, în primul rând, de caracteristica  $n$  a datelor de intrare, fapt accentuat și prin folosirea notațiilor ce reprezintă funcții de argumentul  $n$ .

Aplicarea practică a unui algoritm este posibilă numai atunci când necesarul de memorie și timpul cerut nu încalcă restricțiile impuse de mediul de programare și capacitatea de prelu-crare a calculatorului utilizat.

De exemplu, presupunem că pentru rezolvarea unei probleme există doi algoritmi diferenți, notați prin  $A_1$  și  $A_2$ . Necesarul de memorie și timpul cerut de algoritmul  $A_1$  sunt:

$$\begin{aligned}V_1(n) &= 100n^2 + 4; \\T_1(n) &= n^3 \cdot 10^{-3},\end{aligned}$$

iar de algoritmului  $A_2$ :

$$V_2(n) = 10n + 12;$$

$$T_2(n) = 2^n \cdot 10^{-6}.$$

În aceste formule volumul de memorie se calculează în octeți, iar timpul în secunde.

Necesarul de memorie și timpul cerut de algoritmii  $A_1$ ,  $A_2$  pentru diferite valori ale lui  $n$  sunt prezentate în *tabelul 2.1*.

*Tabelul 2.1. Complexitatea algoritmilor  $A_1$  și  $A_2$*

$n$	10	20	30	40	50
$V_1(n)$	9,77 Kocteți	39,07 Kocteți	87,89 Kocteți	156,25 Kocteți	244,14 Kocteți
$V_2(n)$	112 octeți	212 octeți	312 octeți	412 octeți	512 octeți
$T_1(n)$	1 secundă	8 secunde	27 secunde	64 secunde	125 secunde
$T_2(n)$	0,001 secunde	1,05 secunde	≈18 minute	≈13 zile	≈36 ani

Din *tabelul 2.1* se observă că algoritmul  $A_2$  devine practic inutilizabil pentru datele de intrare caracteristica cărora  $n > 30$ . Pentru astfel de date timpul de execuție a algoritmului  $A_1$  este mult mai mic, însă necesarul de memorie  $V_1(n)$  poate depăși limita impusă de mediul de programare (de exemplu, 64 Kocteți pentru variabilele din programele implementate în mediul de programare Turbo PASCAL).

Determinarea necesarului de memorie  $V(n)$  și a timpului de execuție  $T(n)$  prezintă un interes deosebit la etapa de elaborare a algoritmilor și a programelor respective. Evident, anume pe parcursul acestei etape pot fi eliminată din start acei algoritmi care necesită memorii prea mari sau care au un timp de execuție inaceptabil.

Mentionăm că existența unor calculatoare cu memorii din ce în ce mai performante fac ca atenția informaticienilor să fie îndreptată în special asupra necesarului de timp sau, cu alte cuvinte, asupra **complexității temporale** a algoritmilor.

## Întrebări și exerciții

- 1 **Sistematizează-ți cunoștințele!** Explicați termenul *complexitatea algoritmului*. Numiți indicatorii ce caracterizează complexitatea algoritmilor.
- 2 **Analizează!** Care factori influențează complexitatea unui algoritm?
- 3 **Analizează și aplică!** De ce depinde necesarul de memorie și timpul cerut de un algoritm? Când este posibilă aplicarea practică a unui algoritm?
- 4 **Cercetează!** Algoritmii  $A_1$  și  $A_2$  (vezi *tabelul 2.1*) vor derula în unul dintre mediile de programare Turbo PASCAL, Free Pascal, C/C++. Cum credeți, care algoritm trebuie utilizat în cazul datelor de intrare cu caracteristica: a)  $n = 10$ ; b)  $n = 20$ ; c)  $n = 30$ ? Pentru care valori ale lui  $n$  algoritm  $A_1$  poate fi utilizat în mediul de programare cu care lucrați dvs.?

5

**Cercetează!**

Complexitatea unui algoritm, notat prin  $A_3$ , se caracterizează prin:

$$V_3(n) = 600n^3 + 18;$$

$$T_3(n) = 3^n \cdot 10^{-2}.$$

Cum credeți, pentru ce valori ale lui  $n$  algoritmul  $A_3$  poate derula în mediul de programare cu care lucrăți dvs.?

6

**Integrează cunoștințele și aplică!**

Determinați mărimea datelor de intrare a celor mai reprezentativi algoritmi elaborați de dvs. în procesul studierii limbajelor de programare de nivel înalt.

**2.2****Estimarea necesarului de memorie**

Evaluarea necesarului de memorie  $V(n)$  poate fi făcută însumând numărul de octeți alocăți pentru fiecare variabilă din program. Numărul de octeți alocat unei variabile nestructurate depinde de implementarea limbajului. În cazul mediilor de programare frecvent utilizate, memoria se alocă conform *tabelului 2.2*.

*Tabelul 2.2. Alocarea memoriei interne*

Tipul variabilei PASCAL (C/C++)	Numărul de octeți		
	Turbo PASCAL	Free PASCAL	C/C++
Integer (int)	2	2 sau 4, în funcție de sistemul de operare	2 sau 4, în funcție de sistemul de operare
Longint (int)	4	4	8
Real (float)	6	4 sau 8, în funcție de sistemul de operare	4/8
Boolean (bool)	1	1	-/1
Char (char)	1	1	1
Enumerare (enum)	1	1 sau 2, în funcție de numărul de valori	4
Subdomeniu, doar în PASCAL	conform tipului de bază	conform tipului de bază	-
Referință (&)	4	4 sau 8, în funcție de sistemul de operare	4 sau 8, în funcție de sistemul de operare
Pointer (*)	4	4 sau 8, în funcție de sistemul de operare	Nu este fixat, depinde de tipul datelor referite

În cazul tipurilor structurate de date volumul de memorie necesar unei variabile se calculează însumând numărul de octeți alocăți pentru fiecare componentă.

## Pascal

De exemplu, necesarul de memorie pentru variabilele A, B, p și s din declarațiile Turbo PASCAL

```
var A : array[1..n, 1..n] of real;
      B : array[1..n] of integer;
      p : boolean;
      s : string[10];
```

este

$$V(n) = 6n^2 + 2n + 1 + 10 \text{ (octeți).}$$

## C++

Necesarul de memorie pentru variabilele A, B, p și s din declarațiile C/C++

```
double A[n][n];
long long B[n];
bool p;
char s[10];
```

este

$$V(n) = 8n^2 + 8n + 1 + 10 \text{ (octeți).}$$

În general, necesarul de memorie al unui program depinde nu numai de tipul variabilelor utilizate, dar și de modul de gestionare a memoriei interne a calculatorului. Modalitățile respective se studiază în cursurile avansate de informatică.

## Întrebări și exerciții

- 1 **Explorează!** Determinați cu ajutorul sistemului de asistență al mediului de programare cu care lucrați dvs. necesarul de memorie pentru variabilele structurate și nestructurate.
- 2 **Explică!** Cum se evaluatează volumul de memorie internă necesar pentru înmagazinarea datelor unui algoritm?
- 3 **Învață să înveți!** Aflați cu ajutorul sistemului de asistență al mediului de programare cu care lucrați dvs. modul în care este gestionată memoria internă în procesul execuției unui program.
- 4 **Învață să înveți!** Determinați cu ajutorul sistemului de asistență cum poate fi modificat volumul de memorie internă alocat execuției unui program?

5

**Aplică!**

Calculați necesarul de memorie pentru variabilele din următoarele declarații:

**Pascal**

- a) `var A : array[1..n, 1..n] of integer;  
 B : string;  
 C : array [1..n, 1..n, 1..n] of boolean;`
- b) `type Vector = array[1..n] of real;  
 Matrice = array[1..n] of Vector;  
var A, B, C : Matrice;  
 D : Vector;`
- c) `type Elev = record  
 Nume : string;  
 Prenume : string;  
 NotaMedie : real  
 end;  
 ListaElevi = array[1..n] of Elev;  
var A, B, C : ListaElevi;`
- d) `type Angajat = record  
 NumePrenume : string;  
 ZileLucrare : 1..31;  
 PlataPeZi : real;  
 PlataPeLuna : real  
 end;  
 ListaDePlata = array[1..n] of Angajat;  
var L1, L2 : ListaDePlata;`

**C++**

- a) `int A [n][n];  
string B[256];  
bool C [n][n][n];`
- b) `double A[n][n], B[n][n], C[n][n];  
float D[n];`
- c) `struct Elev {  
 string Nume;  
 string Prenume;  
 float NotaMedie;  
 } A[n], B[n], C[n];`

```
d) struct Angajat {
    char NumePrenume[ 20 ];
    int ZileLucrare;
    float PlataPeZi;
    float PlataPeLuna;
} L1[n], L2[n];
```

- 6 Experimentează!** Evaluați necesarul de memorie pentru programul ce urmează. Compilați acest program pentru următoarele valori consecutive ale constantei  $n$ : 10, 100, 1000, 10000 și.a.m.d. Explicați mesajele afișate la ecran.

### Pascal

```
Program P146;
    { Dimensiunile segmentului date }
const n = 50;
type Matrice = array[1..n, 1..n] of real;
var A, B : Matrice;
begin
    {...introducerea datelor...}
    {...prelucrarea matricelor A si B...}
    writeln('Sfârșit');
    readln;
end.
```

### C++

```
// Program P146
// Dimensiunile segmentului date
#include <iostream>
using namespace std;
#define n 50
float A[n][n], B[n][n];
int main()
{
    //...introducerea datelor...
    //...prelucrarea matricelor A si B...
    cout << "Sfârșit";
}
```

- 7 Analizează și aplică!** Se consideră următorul program:

### Pascal

```
Program P147;
    { Dimensiunile stivei }
var n : integer;

function S(n:integer):real;
begin
    if n=0 then S:=0
        else S:=S(n-1)+n;
end; { S }
```

```

begin
    write('n='); readln(n);
    writeln('s=' , S(n));
    readln;
end.

```

### C++

```

// Program P147
// Dimensiunile stivei
#include <iostream>
using namespace std;
long long n;

double S(long long n)
{
    if (n == 0) return 0;
    else return S(n - 1) + n;
}

int main()
{
    do
    {
        cout << "n = " ; cin >> n;
        cout << "n = " << n << " s = " << S(n) << endl;
    } while (n);
}

```

În acest program suma

$$S(n) = 0 + 1 + 2 + \dots + n$$

este calculată cu ajutorul funcției recursive

$$S(n) = \begin{cases} 0, & \text{dacă } n = 0; \\ S(n-1) + n, & \text{dacă } n > 0. \end{cases}$$

Estimați necesarul de memorie al programului P147. Determinați valoarea maximală a lui  $n$  pentru care programul va derula fără erori. Se consideră că mediul de programare va oferi acestui program 32 Kocetă de memorie operativă.

8

**Experimentează!** Se consideră programul P147 din exercițiul precedent. Atribuindu-i lui  $n$  valorile 10, 100, 1000 10000 ș.a.m.d., determinați pentru care dintre ele acest program derulează fără erori. Estimați volumul de memorie alocat acestui program de mediul de programare cu care lucrați dvs.

9

**Cercetează!** Determinați necesarul de memorie al programului ce urmează. Atribuindu-i lui  $n$  valorile 10, 100, 1000 10000 ș.a.m.d., determinați pentru care valori ale lui  $n$  acest program derulează fără erori. Comparați necesarul de memorie cu cel alocat de mediul de programare cu care lucrați dvs. Explicați mesajele afișate la ecran.

**Pascal**

```
Program P148;
{ Dimensiunile heap-ului }
type Vector = array[1..100] of real;
var p : ^Vector;
    i, n : longint;
begin
    write('n='); readln(n);
    writeln('Start');
    for i:=1 to n do new(p);
    writeln('Sfârșit');
    readln;
end.
```

**C++**

```
// Program P148
// Dimensiunile heap-ului
#include <iostream>
using namespace std;
int main()
{
    double *p = new (nothrow) double[1000000];
    long long i = 0;
    do
    {
        double* p = new(nothrow) double[1000000];
        if (!p) { cout << "Imposibil de alocat memorie\n";
            break;
        }
    } while (p);
}
```

## 2.3 Măsurarea timpului de execuție

În cazul programelor deja elaborate timpul  $T(n)$  cerut de un algoritm poate fi aflat prin măsurări directe.

**Pascal**

În cazul limbajului PASCAL, pentru măsurarea timpului de execuție vom folosi unitatea de program U7 :

```
Unit U7; { Măsurarea timpului }
interface
function TimpulCurent : real;
implementation
uses Dos;
var ore : word;
    minute : word;
    secunde : word;
    sutimi : word;
```

```

function TimpulCurent;
  { Returneaza timpul curent in secunde }
begin
GetTime(ore, minute, secunde, sutimi);
TimpulCurent:=3600.0*ore+60.0*minute+
           1.0*secunde+0.01*sutimi;
end; { TimpulCurent }
end.

```

*Notă. Unitățile de program (texte PASCAL care încep cu cuvântul-cheie **unit**) pot fi compilate separat și conectate la orice program PASCAL cu ajutorul clauzei **uses** (utilizează).*

Unitatea de program U7 oferă programatorului funcția TimpulCurent care returnează o valoare de tip real – timpul în secunde și sutimi de secundă. Indicațiile ceasului de sistem în ore, minute, secunde și sutimi de secundă se citesc cu ajutorul procedurii GetTime din unitatea de program DOS a mediilor de programare PASCAL.

## C++

În cazul limbajului C/C++ pentru măsurarea timpului se folosesc tipurile de date și funcțiile din librăria `<sys/time.h>`, care, evident, trebuie inclusă în programul în curs de elaborare:

```

#include <sys/time.h>
using namespace std;

struct timeval start, stop;
double secunde;

double timpSecunde (struct timeval start, struct timeval stop)
{
    return (double)(stop.tv_usec - start.tv_usec) / 1000000 +
           (double)(stop.tv_sec - start.tv_sec);
}

```

Pentru a înțelege modul în care funcțiile respective se utilizează pentru măsurarea timpului de execuție a unui subprogram, vom analiza programul P149:

## Pascal

```

Program P149; { Timpul de executie a procedurii Sortare }
uses U7;type Vector = array[1..100000] of real;
      var   A : Vector;
      i, n : longint;
      T1, T2 : real; { timpul in secunde }

procedure Sortare(var A:Vector; n:longint);
  { Sortarea elementelor vectorului A }
var i, j : longint;
    r : real;

```

```

begin
  for i:=1 to n do
    for j:=1 to n-1 do
      if A[j]>A[j+1] then
        begin
          r:=A[j];
          A[j]:=A[j+1];
          A[j+1]:=r;
        end;
  end; { Sortare }

begin
  write('Dati numarul de elemente n=');
  readln(n);

{ atribuim lui A valoarea (n, n-1, ..., 3, 2, 1) }
for i:=1 to n do A[i]:=n-i+1;

T1:=TimpulCurent;
Sortare(A, n);
T2:=TimpulCurent;

writeln('Durata de executie', (T2-T1):7:2, ' secunde');
readln;
end.

```

### C++

```

// Program P149;
// Timpul de executie a functiei fara tip Sortare

#include <iostream>
#include <sys/time.h>
using namespace std;

int A[10000], n;
struct timeval start, stop;
double secunde;

double timpSecunde (struct timeval start, struct timeval stop)
{
    return (double)(stop.tv_usec - start.tv_usec) / 1000000 +
           (double)(stop.tv_sec - start.tv_sec);
}

void Sortare (int *x, int n) {
// Sortarea elementelor tabloului x
    int r ;

```

```

for (int i = 1; i < n; i++)
    for (int j = 1; j < n; j++)
        if (x[j] > x[j + 1])
            { r = x[j];
              x[j] = x[j + 1];
              x[j + 1] = r;
            }
}
int main()
{
    cout << "Numarul de elemente "; cin >> n;
    // atribuim lui A valoarea (n, n-1, ..., 3, 2, 1)
    for (int i = 1; i <= n; i++)
        A[i] = n - i + 1;
    gettimeofday(&start, NULL);
    Sortare (A, n);
    gettimeofday(&stop, NULL);
    cout << "Durata de executie - " << timpSecunde(start, stop)
<< "secunde";
    return 0;
}

```

Subprogramul **Sortare** ordonează elementele vectorului A prin metoda bulelor. În această metodă vectorul A este parcurs de  $n$  ori, la fiecare parcurgere efectuându-se  $n-1$  comparații ale elementelor vecine  $A[j]$  și  $A[j+1]$ . Dacă  $A[j] > A[j+1]$ , elementele vecine își schimbă locul.

Pentru a evita introducerea de la tastatură a unui număr mare de date, în programul P149 vectorului A i se atribuie valoarea inițială

$$A = (n, n-1, n-2, \dots, 3, 2, 1).$$

De exemplu, pentru  $n = 4$ , vectorul inițial va fi

$$A = (4, 3, 2, 1).$$

În procesul sortării avem:

$$i = 1, A = (3, 2, 1, 4);$$

$$i = 2, A = (2, 1, 3, 4);$$

$$i = 3, A = (1, 2, 3, 4);$$

$$i = 4, A = (1, 2, 3, 4).$$

Timpul de execuție a subprogramului **Sortare** în cazul unui calculator cu frecvență ceasului de sistem  $1,6\text{ GHz}$  este prezentat în *tabelul 2.3*, iar graficul respectiv – în *figura 2.1*.

*Tabelul 2.3. Timpul de execuție a procedurii **Sortare***

<b>n</b>	10000	20000	30000	40000	50000	60000	70000	80000	90000	100000
<b>T(n), s</b>	2,56	10,23	22,35	40,21	59,45	91,12	123,11	160,19	203,92	248,77

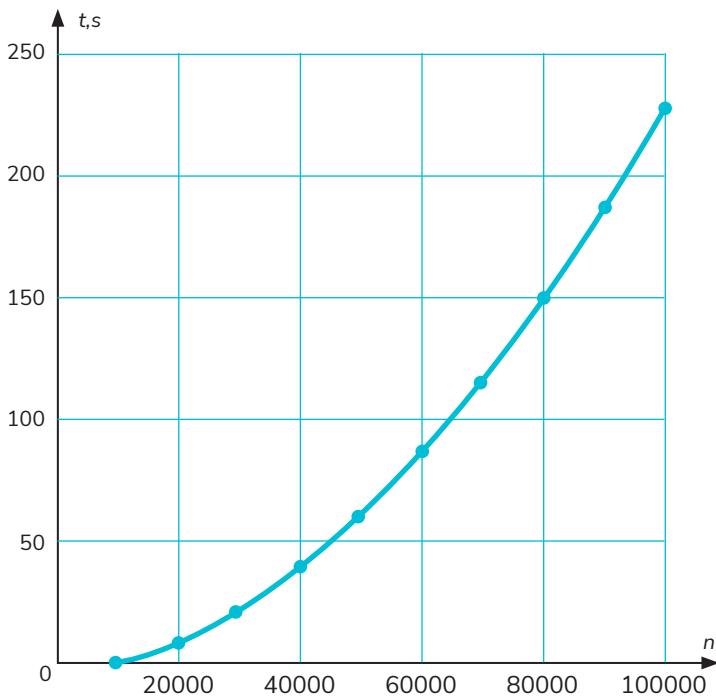


Fig. 2.1. Timpul de execuție a subprogramului Sortare

Accentuăm faptul că sistemele moderne de operare execută în paralel (concomitent) mai multe programe. Pentru a nu include în timpul de execuție a unui subprogram timpul consumat de programele ce se execută în paralel, ele trebuie închise.

Însă unele programe, în special cele din compoziția sistemului de operare, nu pot fi închise. De exemplu, nu pot fi închise programele-pilot ale ecranului și ale tastaturii. În consecință, rezultatele măsurărilor timpilor de execuție conțin anumite erori.

Pentru a micșora erorile de măsurare a timpilor de execuție:

1. Închideți editoarele de texte, procesoarele de foi de calcul, editoarele de prezentări electronice, navigatoarele pe Internet, programele de mesagerie instantă, jocurile didactice și.a.m.d.
2. Compilați programul pentru care veți face măsurări și stocați-l pe disc în formatul cod-obiect.
3. Închideți mediul de programare.
4. Lansați în execuție programul compilat (codul-obiect).
5. Selectați datele de intrare în aşa mod, încât timpul de execuție să depășească câteva sume de secundă.

### Nu uitați!

Timpul de execuție a oricărui subprogram depinde:

- de tipul calculatorului (frecvența ceasului de sistem, numărul de nuclee ale microprocesorului, volumul memoriei-tampon, volumul memoriei operative și.a.);
- de tipul sistemului de operare și de configurațiile acestuia;
- de tipul mediul de programare și de configurațiile acestuia.

În consecință, timpul de execuție a unuia și aceluiași subprogram poate varia de la un calculator la altul.

## Întrebări și exerciții

- 1 **Analizează!** Ce legătură există între timpul necesar execuției unui program și frecvența ceașului de sistem, numărul de nuclee ale procesorului, volumul memoriei operative?
- 2 **Experimentează!** Măsurăți timpul de execuție a procedurii Sortare (vezi programul P149) în cazul calculatorului cu care lucrați dvs. Construiți un grafic similar celui din figura 2.1.
- 3 **Cercetează!** Reprezentați grafic pe un singur desen timpul de execuție a subprogramelor ce urmează.

a)

### Pascal

```
procedure N2(n : longint);
var i, j, q : longint;
    r : real;
begin
    for i:=1 to n do
        for j:=1 to n do
            for q:=1 to 300000 do
                r:=1.0;
end; { N2 }
```

### C++

```
void N2(long long n)
{
    long long i, j, q;
    double r;
    for (i = 1; i <= n; i++)
        for (j = 1; j <= n; j++)
            for (q = 1; q <= 300000;
                q++)
                r = 1.0;
} // N2
```

b)

### Pascal

```
procedure N3(n : longint);
var i, j, k, q : longint;
    r : real;
begin
    for i:=1 to n do
        for j:=1 to n do
            for k:=1 to n do
                for q:=1 to 10000
                    do r:=1.0;
end; { N3 }
```

### C++

```
void N3(long long n)
{
    long long i, j, k, q;
    double r;
    for (i = 1; i <= n; i++)
        for (j = 1; j <= n; j++)
            for (k = 1; k <= n; k++)
                for (q = 1; q <= 10000;
                    q++)
                    r = 1.0;
} // N3
```

c)

### Pascal

```
procedure N4(n : longint);
var i, j, k, m, q : longint;
    r : real;
begin
    for i:=1 to n do
        for j:=1 to n do
            for k:=1 to n do
                for m:=1 to n do
                    for q:=1 to
                        400 do r:=1.0;
end; { N4 }
```

### C++

```
void N4(long long n)
{
    long long i, j, k, m, q;
    double r;
    for (i = 1; i <= n; i++)
        for (j = 1; j <= n; j++)
            for (k = 1; k <= n; k++)
                for (m = 1; m <= n; m++)
                    for (q = 1; q <= 400;
                        q++)
                        r = 1.0;
} // N4
```

Pentru exemplificare, în figura 2.2 sunt prezentate graficele respective în cazul unui calculator cu frecvență ceasului de sistem 1,6 GHz, două nuclee, 6 Gocteți de memorie operativă, sistemul de operare Windows 10.

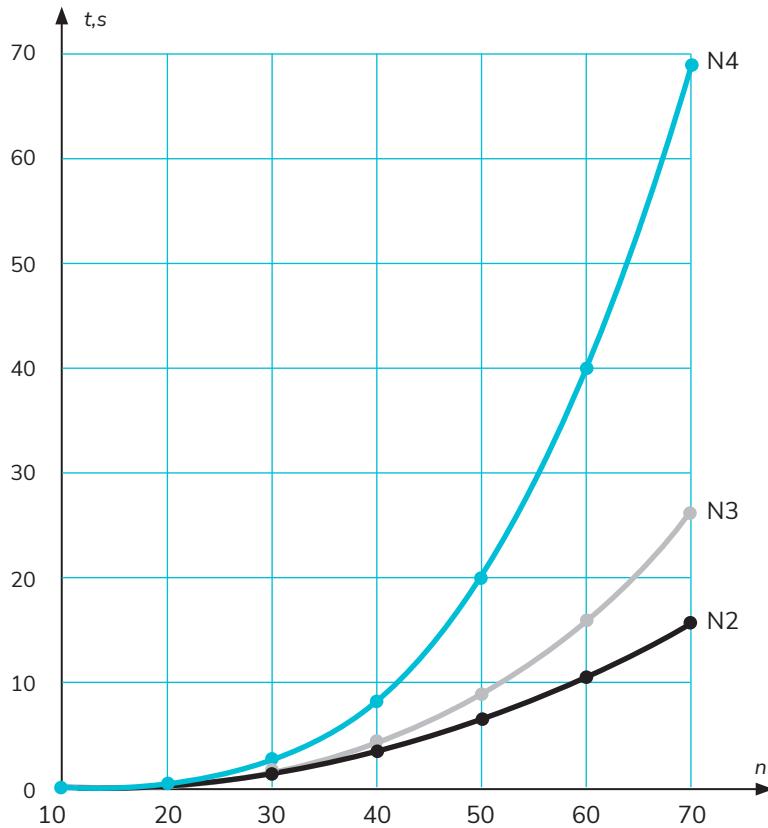


Fig. 2.2. Timpul de execuție a procedurilor N2, N3 și N4

4

**Învață să înveți!** Care este precizia de măsurare a timpului de execuție cu ajutorul funcțiilor oferite de mediul de programare cu care lucrați dumneavoastră? Argumentați răspunsul.

## 2.4 Estimarea timpului cerut de algoritm

În procesul implementării practice a oricărui algoritm apare necesitatea estimării timpului de execuție  $T(n)$  până la testarea și depanarea definitivă a programului ce-l realizează. Cu regret, prin metode teoretice este foarte greu de determinat o expresie exactă pentru  $T(n)$ . Din aceste motive se caută o limită superioară a timpului cerut de algoritm, analizând-se doar cazurile cele mai defavorabile.

Presupunem, în scopuri didactice, că execuția oricărui operator necesită cel mult  $\Delta$  unități de timp. Același timp  $\Delta$  este necesar pentru indexarea componentelor unui tablou, pentru atribuire, pentru instrucțiunea de salt necondiționat. Valoarea concretă a mărimii  $\Delta$  depinde de mediul de programare, capacitatea de prelucrare a calculatorului utilizat și este de ordinul  $10^{-9}$  secunde. În continuare vom estima timpul  $T(n)$  în forma

$$T(n) = Q(n) \cdot \Delta,$$

unde  $Q(n)$  este numărul de operații elementare – adunarea, scăderea, înmulțirea, compararea etc. – necesare pentru soluționarea unei probleme.

Admitem că într-o expresie  $E$  apar  $m$  operatori și  $k$  apeluri ale funcției  $F$ . Evident, numărul  $Q_E$  de operații elementare necesare pentru calcularea expresiei  $E$  se determină ca

$$Q_E = m + k Q_F,$$

unde  $Q_F$  este numărul de operații elementare necesare pentru calcularea funcției  $F$ .

**Exemple:**

	Expresia $E$	Numărul de operații elementare $Q_E$
a)	$a*b+c$	2
b)	$(a < b) \text{or} (c > d)$	3
c)	$\sin(x) + \cos(y)$	$1 + Q_{\sin} + Q_{\cos}$
d)	$a+M[i]$	2
e)	$\sin(x+y) + \sin(x-y)$	$3 + 2Q_{\sin}$

Numărul de operații elementare  $Q_I$  necesare pentru execuția unei instrucțiuni se estimează conform formulelor din *tabelul 2.4*.

Tabelul 2.4. Numărul de operații elementare necesare pentru execuția unei instrucțiuni

Nr. crt.	Instrucțiunea PASCAL Instrucțiunea C / C++	Numărul de operații elementare
1.	Atribuirea $v := E$ $v = E$	$Q_E + 1$
2.	Apelul procedurii $P$ Apelul funcției fără tip $P$	$Q_P + 1$
3.	Selectie <b>if</b> $E$ <b>then</b> $I_1$ <b>else</b> $I_2$ <b>if</b> $(E)$ $I_1$ ; <b>else</b> $I_2$	$Q_E + \max \{Q_{I_1}, Q_{I_2}\} + 1$
4.	Selectie multiplă <b>case</b> $E$ <b>of</b> $I_1; I_2; \dots; I_k$ <b>end</b> <b>switch</b> ( $E$ ) { <b>case</b> $I_1$ ; <b>case</b> $I_2$ ; ... <b>case</b> $I_k$ }	$Q_E + \max \{Q_{I_1}, Q_{I_2}, \dots, Q_{I_k}\} + \kappa + 1$
5.	Ciclu cu contor <b>for</b> $v := E_1$ <b>to/downto</b> $E_2$ <b>do</b> $I$ <b>for</b> ( <b>int</b> $v = E_1$ ; $v \leq E_2$ ; $i++$ ) $I$	$Q_{E_1} + Q_{E_2} + mQ_I + m + 1$
6.	Ciclu cu test inițial <b>while</b> $E$ <b>do</b> $I$ <b>while</b> $(E)$ { $I$ ; }	$(m + 1)Q_E + mQ_I + 1$
7.	Ciclu cu test final <b>repeat</b> $I$ <b>until</b> $E$ <b>do</b> { $I$ ; } <b>while</b> $(E)$	$mQ_I + mQ_E + 1$

8.	Instrucția compusă <b>begin</b> $I_1; I_2; \dots; I_k$ <b>end</b> { $I_1; I_2; \dots; I_k$ }	$Q_{I_1} + Q_{I_2} + \dots + Q_{I_k} + 1$
9.	Instrucția <b>with</b> $v$ <b>do</b> $I$ Nu există în C/C++	$Q_I + 1$
10.	Saltul <b>goto</b> <b>goto</b>	1

Formulele din *tabelul 2.4* pot fi deduse urmând modul de execuție a fiecărei instrucții. În acest tabel  $v$  reprezintă o variabilă sau un nume de funcție,  $E$  – o expresie, iar  $I$  – o instrucție. Numărul de execuții ale instrucției  $I$  din cadrul unui ciclu este notat prin  $m$ . Menționăm că ciclurile unui program pot fi organizate și cu ajutorul instrucțiunilor de salt condiționat și necondiționat, însă o astfel de utilizare a acestor instrucții contravine regulilor programării structurate.

Pentru exemplificare, vom estima numărul de operații elementare  $Q(n)$  necesare ordonării elementelor unui vector prin metoda bulelor:

### Pascal

```
procedure Sortare(var A:Vector; n:integer);
var i, j : integer;
    r : real;
{1} begin
{2} for i:=1 to n do
{3}     for j:=1 to n-1 do
{4}         if A[j]>A[j+1] then
{5}             begin
{6}                 r:=A[j];
{7}                 A[j]:=A[j+1];
{8}                 A[j+1]:=r;
            end;
{end; { Sortare }}
```

### C++

```
void Sortare(double *A, int n)
{
    for (int i = 1; i < n; i++)          // 1
        for (int j = 1; j < n; j++)      // 2
            if (A[j] > A[j + 1])        // 3
            {
                r = A[j];               // 4
                A[j] = A[j + 1];         // 5
                A[j + 1] = r;           // 6
            }
    return;
}
```

Instrucțiunile  $I_1, I_2, \dots, I_8$  ale subprogramului **Sortare** vor fi referite cu ajutorul comentariilor din liniile respective de program. Prin  $Q_j$  vom nota numărul de operații elementare necesare pentru executarea instrucțiunii  $I_j$ . În conformitate cu *tabelul 2.4*, obținem:

$$Q_6 = 2;$$

$$Q_7 = 4;$$

$$Q_8 = 3;$$

$$Q_5 = Q_6 + Q_7 + Q_8 + 1 = 10;$$

$$Q_4 = 4 + Q_5 + 1 = 15;$$

$$Q_3 = 0 + 1 + (n-1)Q_4 + (n-1) + 1 = 16n - 14;$$

$$Q_2 = 0 + 0 + nQ_3 + n + 1 = 16n^2 - 13n + 1;$$

$$Q_1 = Q_2 + 1 = 16n^2 - 13n + 2.$$

Prin urmare, numărul de operații elementare

$$Q(n) = 16n^2 - 13n + 2,$$

iar timpul cerut de procedura **Sortare**

$$T(n) = (16n^2 - 13n + 2)\Delta.$$

Din exemplul studiat mai sus se observă că ordinea parcurgerii instrucțiunilor este impusă de structura programelor respective. Evident, mai întâi se analizează instrucțiunile simple, iar apoi cele structurate. În cazul instrucțiunilor imbricate mai întâi se analizează instrucțiunile din interior, apoi cele care le cuprind.

Expresiile analitice  $T(n)$  obținute în urma analizei programelor scrise într-un limbaj de programare de nivel înalt pot fi folosite pentru determinarea experimentală a timpului  $\Delta$  necesar efectuării unei operații elementare.

De exemplu, pentru subprogramul **Sortare** (vezi *tabelul 2.3*)  $n = 10000$  și  $T(n) = 2,56$  s.

Din ecuația

$$(16n^2 - 13n + 2)\Delta = 2,56$$

obținem  $\Delta \approx 1,6 \cdot 10^{-9}$  secunde.

Evident, această valoare este valabilă numai pentru un calculator cu frecvența ceasului de sistem 1,6 GHz, utilizat în acest manual pentru a măsura timpul de execuție a procedurii respective.

De exemplu, în cazul unui calculator cu frecvența ceasului de sistem 3,2 GHz se obține valoarea  $\Delta \approx 0,8 \cdot 10^{-9}$  secunde.

## Întrebări și exerciții

- 1 Experimentează!** Determinați cu ajutorul programului P149 valoarea  $\Delta$  pentru mediul de programare și calculatorul cu care lucrați dvs.
- 2 Exersează!** Determinați numărul de operații elementare  $Q_j$  necesare pentru execuția următoarelor instrucțiuni:

### Pascal

a) `x:=2*a-6*(y+z);`

b) `p:=not(a=b)and(c>d);`

- c) `p:=(a in R)and(b in P);`
- d) `if a>b then x:=0 else x:=a+b;`
- e) `case i of  
 1: x:=0;  
 2: x:=a+b;  
 3: x:=a+b+c;  
end;`
- f) `for i:=1 to n do A[i]:=2*A[i];`
- g) `for i:=1 to n do A[i]:=B[i+1]-C[i-2];`
- h) `i:=0; while i<n do begin i:=i+1 end;`
- i) `i:=n; repeat i:=i-1 until i=0;`
- j) `begin i:=0; s:=0; r:=0 end;`
- k) `with A do begin x:=0; y:=0 end.`

**C++**

- a) `x = 2 * a - 6 * (y + z);`
- b) `p = !(a == b) && (c > d);`
- c) `set <int> p; set <int> R;  
p = (r.count(a) == 1) && (r.count(b) == 1);`
- d) `if (a > b) x = 0; else x = a + b;`
- e) `switch(i)  
{  
 case 1: x = 0;  
 case 2: x = a + b;  
 case 3: x = a + b + c;  
}`
- f) `for (int i = 1; i<= n; i++) A[i] = 2 * A[i];`
- g) `for (int i = 1; i<= n; i++) A[i] = B[i + 1] - C[i - 2];`

- h) `i = 0; while (i < n) { i++; }`
- i) `i = n; do {i--;} while (i != 0);`
- j) `{ i = 0; s = 0; r = 0; }`
- k) `{ A.x = 0; A.y = 0; }`

**3 Aplică!** Estimați numărul operațiilor elementare  $Q(n)$  din subprogramele N2, N3 și N4 (vezi exercițiul 3 din paragraful precedent).

**4 Analizează!** În cazul subprogramului Sortare pentru un calculator cu frecvența ceasului de sistem  $f_1 = 1,6 \text{ GHz}$  s-a obținut  $\Delta_1 \approx 1,6 \cdot 10^{-9} \text{ s}$ . Pentru același tip de calculator, însă cu frecvența ceasului de sistem  $f_2 = 3,2 \text{ GHz}$ , s-a obținut  $\Delta_2 \approx 0,8 \cdot 10^{-9} \text{ s}$ . Se observă că

$$\frac{f_2}{f_1} = \frac{3,2 \cdot 10^9}{1,6 \cdot 10^9} = 2 \quad \text{și} \quad \frac{\Delta_2}{\Delta_1} = \frac{0,8 \cdot 10^{-9}}{1,6 \cdot 10^{-9}} = \frac{1}{2}.$$

Prin ce se explică acest fapt?

**5 Elaborează!** În procesul compilării instrucțiunile din limbajele de nivel înalt sunt translate în una sau mai multe instrucțiuni din limbajul cod-mașină. Numărul concret de instrucțiuni din limbajul cod-mașină depinde de mediul de programare și tipul calculatorului utilizat. Elaboreazăți planul unui experiment care ne-ar permite să estimăm numărul de instrucțiuni cod-mașină în care este translatată fiecare instrucțiune a limbajului de programare de nivel înalt pe care îl studiați.

**6 Experimentează!** E cunoscut faptul că timpul de execuție a instrucțiunilor din limbajul cod-mașină depinde de tipul lor. De exemplu, o instrucțiune care adună două numere întregi este mai rapidă decât o instrucțiune care adună două numere reale. În consecință, și valorile  $\Delta$ , determinate prin măsurarea timpului de execuție a programelor scrise în limbaje de programare de nivel înalt, de asemenea depind de tipul datelor utilizate. Verificați experimental această afirmație în cazul calculatorului cu care lucrați dvs.

**7 Cercetează!** Capacitatea de prelucrare a unui calculator se măsoară în Gips – Gigainstrucțiuni pe secundă ( $10^9$  instrucțiuni pe secundă). Pentru a măsura această caracteristică, producătorii de calculatoare utilizează instrucțiunile limbajului cod-mașină. De exemplu, capacitatea de prelucrare a calculatoarelor personale moderne este de 1–5 Gips. Pentru un programator însă ar prezenta interes nu doar capacitatea de prelucrare, exprimată în instrucțiuni ale limbajului cod-mașină, dar și în instrucțiuni ale limbajelor de programare de nivel înalt. Elaboreazăți planul unui experiment care ar permite estimarea acestei caracteristici pentru calculatorul cu care lucrați dvs.

## 2.5 Complexitatea temporală a algoritmilor

În informatică complexitatea temporală a algoritmilor se caracterizează prin timpul de execuție  $T(n)$  sau numărul de operații elementare  $Q(n)$ . Întrucât calculatoarele moderne au o viteză de calcul foarte mare –  $10^9 \dots 10^{11}$  instrucțiuni pe secundă, problema timpului de execuție se pune numai pentru valorile mari ale lui  $n$ . În consecință, în formulele ce exprimă numărul de operații elementare  $Q(n)$  prezintă interes numai **termenul dominant**, adică cel care tinde cât mai repede la infinit. Importanța termenului dominant față de ceilalți este pusă în evidență în tabelul 2.5.

Tabelul 2.5. Valorile termenilor dominanți

$n$	$\log_2 n$	$n^2$	$n^3$	$n^4$	$2^n$
2	1	4	8	16	4
4	2	16	64	256	16
8	3	64	512	4096	256
16	4	256	4096	65536	65536
32	5	1024	32768	1048576	4294967296

De exemplu, numărul de operații elementare ale subprogramului Sortare se exprimă prin formula

$$Q(n) = 16n^2 - 13n + 2.$$

Termenul dominant din această formulă este  $16n^2$ . Evident, pentru valorile mari ale lui  $n$  numărul de operații elementare

$$Q(n) \approx 16n^2,$$

iar timpul de execuție

$$T(n) \approx 16n^2\Delta.$$

În funcție de complexitatea temporală, algoritmii se clasifică în:

- algoritmi polinomiali;
- algoritmi exponențiali;
- algoritmi nedeterminist polinomiali.

Un algoritm se numește **polinomial** dacă termenul dominant are forma  $Cn^k$ , adică

$$Q(n) \approx Cn^k; \quad T(n) \approx Cn^k\Delta,$$

unde:

$n$  este caracteristica datelor de intrare;

$C$  – o constantă pozitivă;

$k$  – un număr natural.

Complexitatea temporală a algoritmilor polinomiali este redată prin notația  $O(n^k)$  care se citește „algoritm cu timpul de execuție de ordinul  $n^k$ ”, sau, mai pe scurt, „**algoritm de ordinul  $n^k$** ”. Evident, există algoritmi polinomiali de ordinul  $n$ ,  $n^2$ ,  $n^3$  și.m.d.

De exemplu, algoritmul de sortare a elementelor unui vector prin metoda bulelor este un algoritm polinomial de ordinul  $n^2$ . Acest lucru se observă și din graficul timpului de execuție  $T(n)$  a procedurii Sortare, grafic prezentat în figura 2.1.

Un algoritm se numește **exponențial** dacă termenul dominant are forma  $Ck^n$ , adică

$$Q(n) \approx Ck^n; \quad T(n) \approx Ck^n\Delta,$$

unde  $k > 1$ . Complexitatea temporală a algoritmilor exponențiali este redată prin notația  $O(k^n)$ .

Mentionăm faptul că tot exponențiali se consideră și algoritmii de complexitatea  $n^{\log n}$ , cu toate că această funcție nu este exponențială în sensul strict matematic al acestui cuvânt.

Din comparația vitezei de creștere a funcțiilor exponențială și polinomială (vezi tabelul 2.5) rezultă că algoritmii exponențiali devin inutilizabili chiar pentru valori nu prea mari ale lui  $n$ . Pentru exemplificare, amintim tabelul 2.1 în care este prezentat timpul de execuție  $T_1(n)$  a unui algoritm polinomial de ordinul  $O(n^3)$  și timpul de execuție  $T_2(n)$  a unui algoritm exponențial de ordinul  $O(2^n)$ .

Algoritmii **nederminist polinomiali** se studiază în cursurile avansate de informatică.

În funcție de complexitatea temporală, se consideră că o problemă este **ușor rezolvabilă** dacă pentru soluționarea ei există un algoritm polinomial. O problemă pentru care nu există un algoritm polinomial se numește **dificilă**. Accentuăm faptul că această clasificare se referă doar la analiza asimptotică a algoritmilor, adică la comportarea lor pentru valori mari ale lui  $n$ . Pentru valorile mici ale lui  $n$  situația poate fi diferită.

De exemplu, presupunem că pentru soluționarea unei probleme există doi algoritmi, unul polinomial cu timpul de execuție  $T_1(n)$  și altul exponențial cu timpul de execuție  $T_2(n)$ :

$$T_1(n) = 1000n^2\Delta;$$

$$T_2(n) = 2^n\Delta.$$

Prin calcule directe se poate verifica că pentru  $n = 1, 2, 3, \dots, 18$  timpul  $T_2(n) < T_1(n)$ . Prin urmare, în situația  $n \leq 18$  vom prefera algoritmul exponențial.

În practică, elaborarea programelor de calculator presupune parcurserea următoarelor etape:

- formularea exactă a problemei;
- determinarea complexității temporale a problemei propuse – problemă ușor rezolvabilă sau problemă dificilă;
- elaborarea algoritmului respectiv și implementarea lui pe un sistem de calcul.

Evident, în cazul unor probleme ușor rezolvabile, programatorul va depune toate eforturile pentru a inventa algoritmi polinomiali, adică algoritmi de ordinul  $n^k$ , astfel încât parametrul  $k$  să ia valori cât mai mici. În cazul problemelor dificile se va da prioritate algoritmilor care minimizează timpul de execuție cel puțin pentru datele de intrare frecvent utilizate în aplicațiile practice. Metodele de clasificare a problemelor în probleme ușor rezolvabile și probleme dificile se studiază în cursurile avansate de informatică.

## Întrebări și exerciții

1

**Exersează!**

Indicați termenii dominanți:

- a)  $12n + 5$ ;
- b)  $6n^2 + 100n + 18$ ;
- c)  $15n^3 + 1000n^2 - 25n + 6000$ ;

- d)  $2000n^3 + 2^n + 13;$
- e)  $n^{\log_2 n} + n^5 + 300n^2 + 6;$
- f)  $3^n + 2^n + 14n^3 + 21;$
- g)  $n^5 + 10n^4 + 200n^3 + 300n^2 + 1000n.$

**2 Sistematizează-ți cunoștințele!** Cum se clasifică algoritmi în funcție de complexitatea temporală?

**3 Exersează!** Determinați tipul algoritmilor, cunoscând complexitatea temporală:

- a)  $Q(n) = 200n + 15;$
- b)  $Q(n) = 2^n + 25n^2 + 1000;$
- c)  $Q(n) = n^3 + 3^n + 6000n^2 + 10^6;$
- d)  $Q(n) = 3^n + 2^n + n^{10} + 4000;$
- e)  $Q(n) = n^{\log_2 n} + n^3 + n^2 + 1500;$
- f)  $Q(n) = 100n^2 + 15n^3 + 8^n + 900.$

**4 Explică!** Prin ce se explică importanța termenului dominant în analiza complexității temporale a algoritmilor?

**5 Cercetează!** Determinați ordinul de complexitate temporală a algoritmilor descriși cu ajutorul subprogramelor N2, N3 și N4 (vezi exercițiul 3 din paragraful 2.3). Comparați viteza de creștere a timpilor de execuție  $T_{N2}(n)$ ,  $T_{N3}(n)$  și  $T_{N4}(n)$ , folosind în acest scop graficele din figura 2.2.

**6 Aplică!** Se consideră un algoritm format din  $k$  cicluri imbricate:

### Pascal

```
for i1:=1 to n do
    for i2:=1 to n do
        ...
        for ik:=1 to n do P
```

### C++

```
for ( int i1 = 1; i1 <= n; i1++ )
    for ( int i2 = 1; i2 <= n; i2++ )
        ...
        for ( int ik = 1; ik <= n; ik++ )
```

Numărul de operații elementare  $Q_P$  efectuate în procedura P este o mărime constantă. Estimați complexitatea temporală a algoritmului.

7

**Integrează cunoștințele și aplică!**

Schițați un algoritm pentru rezolvarea următoarei probleme:

Se consideră mulțimea A formată din  $n$  numere întregi. Determinați dacă există cel puțin o submulțime  $B$ ,  $B \subseteq A$ , suma elementelor căreia este egală cu  $m$ . De exemplu, pentru  $A=\{-3, 1, 5, 9\}$  și  $m=7$ , o astfel de submulțime există, și anume  $B=\{-3, 1, 9\}$ .

Estimați complexitatea temporală a algoritmului elaborat.

**2.6****Iterativitate sau recursivitate**

Pe parcursul dezvoltării informaticii s-a stabilit că multe probleme de o reală importanță practică pot fi rezolvate cu ajutorul unor metode standard, denumite **tehnici de programare**: recursia, trierea, metoda reluării, metodele euristice și.a.

Una dintre cele mai răspândite tehnici de programare este **recursia**. Amintim că recursia se definește ca o situație în care un subprogram se autoapeleză fie direct, fie prin intermediul altui subprogram.

Tehnicile în studiu se numesc respectiv **recursia directă** și **recursia indirectă**.

În general, elaborarea unui program recursiv este posibilă numai atunci când se respectă următoarea **regulă de consistență**: soluția problemei trebuie să fie direct calculabilă ori calculabilă cu ajutorul unor valori direct calculabile. Cu alte cuvinte, definirea corectă a unui algoritm recursiv presupune că în procesul derulării calculelor trebuie să existe:

- cazuri elementare, care se rezolvă direct;
- cazuri care nu se rezolvă direct, însă procesul de calcul în mod obligatoriu progresează către un caz elementar.

De exemplu, în definiția recursivă a funcției factorial  $fact: N \rightarrow N$ ,

$$fact(n) = \begin{cases} 1, & \text{dacă } n = 0; \\ n \cdot fact(n-1), & \text{dacă } n > 0, \end{cases}$$

deosebim:

– Cazul elementar  $n = 0$ . În acest caz valoarea  $fact(0)$  este direct calculabilă, și anume  $fact(0)=1$ .

– Cazurile neelementare  $n > 0$ . În astfel de cazuri valorile  $fact(n)$  nu sunt direct calculabile, însă procesul de calcul progresează către cazul elementar  $fact(0)$ .

De exemplu, pentru  $n = 3$  obținem:

$$fact(3) = 3 \cdot fact(2) = 3 \cdot 2 \cdot fact(1) = 3 \cdot 2 \cdot 1 \cdot fact(0) = 3 \cdot 2 \cdot 1 \cdot 1 = 6.$$

Prin urmare, definiția recursivă a funcției  $fact(n)$  este o **definiție consistentă**. Amintim că funcția  $fact(n)$  poate fi exprimată într-un limbaj de programare de nivel înalt, urmând direct definiția în forma:

**Pascal**

```
function Fact(n:Natural):Natural;
begin
  if n=0 then Fact:=1
  else Fact:=n*Fact(n-1)
end;
```

**C++**

```
long Fact(int n)
{
    if (n == 0) return 1;
    else return n * Fact(n - 1);
}
```

În memoria operativă, în cazul apelurilor recursive, valorile curente ale argumentului și ale funcției recursive sunt depuse într-o stivă.

În limbajul cotidian, prin **stivă** (în limba engleză *stack*) înțelegem o mulțime de obiecte de același fel (și cu aceleași dimensiuni), așezate ordonat unele peste altele cu proprietatea că operațiile de introducere și extragere a obiectelor se fac doar în partea de sus a ei. Poziția ocupată în stivă de ultimul element introdus poartă numele de **vârf**. O stivă fără niciun element se numește **stivă vidă**.

În informatică, stiva este realizată printr-o zonă din memoria operativă, cu proprietatea că operațiile de introducere și extragere a datelor se fac doar la un singur capăt al ei.

Modul de gestionare a stivei în cazul apelului `Fact(3)` este prezentat în figura 2.3.

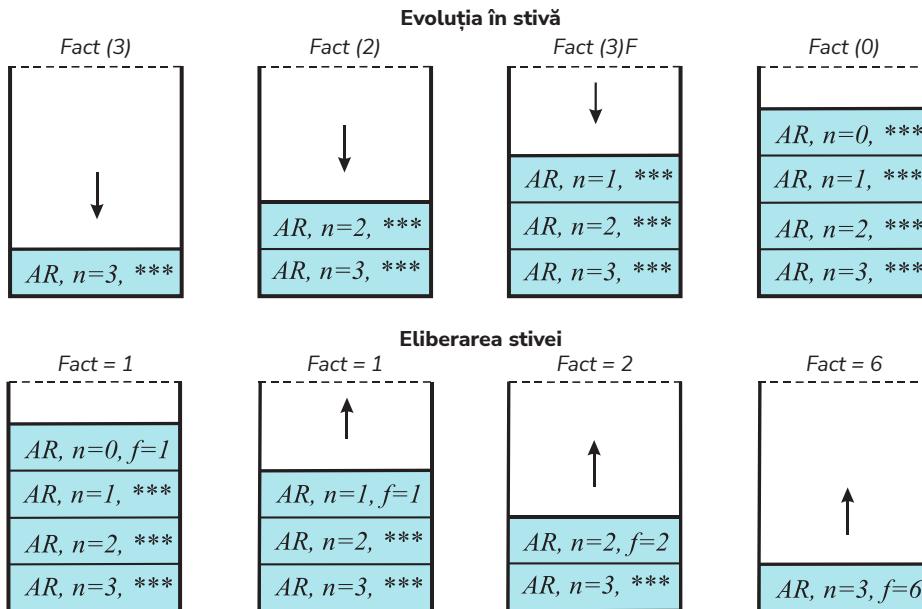


Fig. 2.3. Gestionarea stivei în cazul apelului `Fact(3)`:  
 AR – adresa de revenire; n – valoarea curentă a parametrului actual;  
 \*\*\* – spațiu pentru memorarea valorii f returnate de funcția `Fact`

Într-un mod similar, în definiția recursivă a funcției `incons: N→N`,

$$\text{incons}(n) = \begin{cases} 1, & \text{dacă } n = 0; \\ n \cdot \text{incons}(n+1), & \text{dacă } n > 0, \end{cases}$$

deosebim cazul elementar  $n=0$  și cazurile neelementare  $n>0$ . Însă, spre deosebire de funcția `fact(n)`, pentru  $n > 0$  valorile `incons(n)` nu pot fi calculate, întrucât procesul de calcul progresează către `incons( $\infty$ )`.

De exemplu, pentru  $n=3$  obținem:

$$\text{incons}(3) = 3 \cdot \text{incons}(4) = 3 \cdot 4 \cdot \text{incons}(5) = 3 \cdot 4 \cdot 5 \cdot \text{incons}(6) = \dots$$

Prin urmare, definiția recursivă a funcției  $incons(n)$  este o **definiție inconsistentă** și teoretic procesul de calcul va dura la nesfârșit. În practică, calculele vor fi întrerupte de sistemul de operare în momentul depășirii capacitații de memorare a stivei sau în cazul depășirii capacitații dispozitivului aritmetic.

Accentuăm faptul că mediile actuale de programare nu asigură verificarea consistenței algoritmilor recursivi, acest lucru revenindu-i programatorului.

După cum s-a văzut în capitolele precedente, orice algoritm recursiv poate fi transcris într-un algoritm iterativ și invers. Alegerea tehnicii de programare – **iterativitate** sau **recursivitate** – ține, de asemenea, de competența programatorului. Evident, această alegere trebuie făcută luând în considerare avantajele și neajunsurile fiecărei metode, care variază de la caz la caz. Pentru exemplificare, în *tabelul 2.6.* sunt prezentate rezultatele unui studiu comparativ al ambelor tehnici de programare în cazul prelucrării automate a textelor.

*Tabelul 2.6. Studiul comparativ al iterativității și recursivității (prelucrarea automată a textelor)*

Nr. crt.	Caracteristici	Iterativitate	Recursivitate
1.	Necesarul de memorie	mic	mare
2.	Timpul de execuție		același
3.	Structura programului	complicată	simplă
4.	Volumul de muncă necesar pentru scrierea programului	mare	mic
5.	Testarea și depanarea programelor	simplă	complicată

În general, algoritmii recursivi sunt recomandați în special pentru problemele ale căror rezultate sunt definite prin relații de recurență: analiza sintactică a textelor, prelucrarea datelor structurate, procesarea imaginilor și.a. Un exemplu tipic de astfel de probleme este analiza gramaticală a programelor scrise în limbaje de programare de nivel înalt, sintaxa cărora, după cum se știe, este definită prin relații de recurență.

## Întrebări și exerciții

- 1 **Cercetează!** Explicați termenul *tehnici de programare*.
- 2 **Sistematizează-ți cunoștințele!** Care este diferența dintre *recursia directă* și *recursia indirectă*? Dați exemple.
- 3 **Explică!** Ce condiții trebuie respectate pentru ca definiția unui algoritm recursiv să fie corectă?
- 4 **Analizează!** Care este diferența dintre definițiile recursive consistente și definițiile recursive inconsistentе?

5

**Exersează!** Se consideră următoarele definiții recursive de funcții. Care dintre aceste definiții sunt consistente? Argumentați răspunsul.

- |                           |  |
|---------------------------|--|
| a) $f: N \rightarrow N$ , | $f(n) = \begin{cases} 1, & \text{dacă } n = 0; \\ n + f(n-1), & \text{dacă } n > 0; \end{cases}$                             |
| b) $f: N \rightarrow N$ , | $f(n) = \begin{cases} 1, & \text{dacă } n = 0; \\ n + f(n), & \text{dacă } n > 0; \end{cases}$                               |
| c) $f: Z \rightarrow Z$ , | $f(i) = \begin{cases} 1, & \text{dacă } i = 0; \\ i + f(i-1), & \text{dacă } i \neq 0; \end{cases}$                          |
| d) $f: N \rightarrow N$ , | $f(n) = \begin{cases} 1, & \text{dacă } n = 0; \\ i + g(n-1), & \text{dacă } i > 0; \end{cases}$                             |
| e) $g: N \rightarrow N$ , | $f(n) = \begin{cases} 2, & \text{dacă } n = 0; \\ n + f(n-1), & \text{dacă } n > 0; \end{cases}$                             |
| f) $f: N \rightarrow N$ , | $f(n) = \begin{cases} n, & \text{dacă } n = 0; \\ (n \bmod 10) + f(n \text{ div } 10), & \text{dacă } n \geq 0. \end{cases}$ |

6

**Aplică!** Lansați în execuție programul ce urmează. Explicați mesajele afișate la ecran.

### Pascal

```
Program P150;
{ Depasirea capacitatii de memorare a stivei }
type Natural = 0..Maxint;

function Incons(n:Natural):Natural;
{ Definitie recursiva inconsistenta }
begin
  writeln('Apel recursiv n=', n);
  if n=0 then Incons:=1
  else Incons:=n*Incons(n+1);
end; { Incons }

begin
  writeln(Incons(3));
  readln;
end.
```

### C++

```
// Program P150;
// Depasirea capacitatii de memorare a stivei
// Deoarece volumul de memorie, alocat stivei in C/C++ este net
// superior volumului alocat in Pascal, in functie a fost declarata
// variabila tablou[n], care consuma progresiv memoria
#include <iostream>
using namespace std;
int Incons(int n)
// Definitie recursiva inconsistenta
{
    double tablou[n];
    cout << "Apel recursiv n = "<< n << endl;
    if (!n) return 1;
    else
    {
        for (int i = 1; i <= n; i++) tablou[i] = i;
        return n * Incons(n + 1);
    }
}
```

```

} // Incons

int main()
{
    cout << Incons(3) << endl;
}

```

Reprezentați pe un desen similar celui din figura 2.3 modul de gestionare a stivei în cazul apelului *Incons*(3).

- 7 Analizează!** Suma primelor  $n$  numere naturale  $S(n)$  poate fi calculată cu ajutorul funcției iterative

$$S(n) = 0 + 1 + 2 + \dots + n = \sum_{i=0}^n i$$

sau al funcției recursive

$$S(n) = \begin{cases} 0, & \text{dacă } n = 0; \\ n + S(n - 1), & \text{dacă } n > 0. \end{cases}$$

Utilizând ca model tabelul 2.6, efectuați un studiu comparativ al algoritmilor destinați calculării sumei  $S(n)$ .

- 8 Integrează cunoștințele și aplică!** Se consideră următoarele formule metalingvistice:

*<Cifră>* ::= 0|1|2|3|4|5|6|7|8|9

*<Număr>* ::= *<Cifră>* { *<Cifră>* }

*<Operator>* ::= + | - | \*

*<Expresie>* ::= *<Număr>* | (*<Expresie>*) | *<Expresie>* *<Operator>* *<Expresie>*

Scriți o funcție recursivă care returnează valoarea *true* dacă sirul de caractere *S* este conform definiției unității lexicale *<Expresie>* și *false* în caz contrar.

- 9 Studiu de caz** Scrieți o funcție iterativă care returnează valoarea *true* dacă sirul de caractere *S* este conform definiției unității lexicale *<Expresie>* din exercițiul 8 și *false* în caz contrar. Utilizând ca model tabelul 2.6, efectuați un studiu comparativ al algoritmului iterativ și algoritmului recursiv.

- 10 Programează!** Elaborați un subprogram recursiv care calculează suma cifrelor unui număr natural  $n$ . Examinați cazurile  $n \leq \text{MaxInt}$  și  $n \leq 10^{250}$ .

- 11 Programează!** Imaginele în alb-negru (fig. 2.4) pot fi codificate cu ajutorul unei matrice binare  $B = \{b_{ij}\}_{n \times m}$ ,  $1 \leq n, m \leq 30$ . Elementul  $b_{ij}$  indică culoarea microzonei respective: neagră ( $b_{ij} = 1$ ) sau albă ( $b_{ij} = 0$ ).

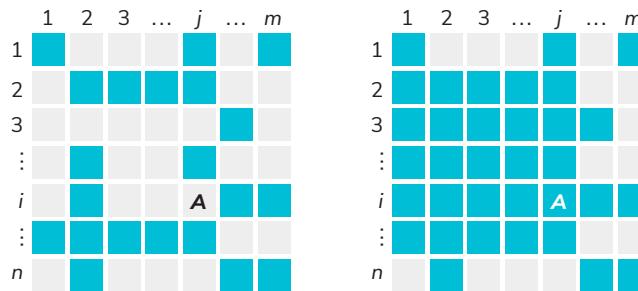


Fig. 2.4. Colorarea unei suprafețe închise: a – imaginea inițială; b – imaginea finală

Elaborați o procedură recursivă pentru colorarea unei suprafețe închise, adică fără intreruperi, specificate prin coordonatele  $(i, j)$  ale oricărei microzone A din interiorul ei.

- 12 Programează!** Elaborați o procedură care determină câte obiecte conține o imagine în alb-negru. Imaginea este împărțită în microzone și codificată cu ajutorul matricei binare  $B = \{b_{ij}\}_{n \times m}$ . Elementul  $b_{ij}$  indică culoarea microzonei cu coordonatele  $(i, j)$ .
- 13 Cercetează!** Efectuați un studiu comparativ al algoritmilor iterativi și algoritmilor recursivi destinați soluționării problemelor din exercițiile 11 și 12.

## 2.7 Metoda trierii

În metoda trierii se presupune că soluția unei probleme poate fi găsită analizând consecutiv elementele  $s_i$  ale unei mulțimi finite

$$S = \{s_1, s_2, \dots, s_i, \dots, s_k\},$$

denumită **mulțimea soluțiilor posibile**. În cele mai simple cazuri elementele  $s_i, s_i \in S$  pot fi reprezentate prin valori aparținând unor tipuri ordinar de date: `integer`, `boolean`, `char`, `enumerare` sau `subdomeniu`. În problemele mai complicate suntem nevoiți să reprezentăm aceste elemente prin tablouri, articole sau mulțimi. Menționăm că în majoritatea problemelor soluțiile posibile  $s_1, s_2, \dots, s_k$  nu sunt indicate explicit în enunțul problemei și elaborarea algoritmilor pentru calcularea lor cade în sarcina programatorului.

**Schema generală** a unui algoritm bazat pe metoda trierii poate fi redată cu ajutorul unui ciclu:

### Pascal

```
for i := 1 to k do
  if SolutiePosibila(si) then PrelucrareaSolutiei(si)
```

### C++

```
for (int i = 1; i <= k; i++)
  if (SolutiePosibila(si)) PrelucrareaSolutiei(si)
```

unde `SolutiePosibila` este o funcție booleană care returnează valoarea `true` dacă elementul  $s_i$  satisfac condițiile problemei și `false` în caz contrar, iar `PrelucrareaSolutiei` este o procedură care efectuează prelucrarea elementului selectat. De obicei, în această procedură soluția  $s_i$  este afișată la ecran.

În continuare vom analiza două exemple care pun în evidență avantajele și neajunsurile algoritmilor bazați pe metoda trierii.

**Exemplul 1.** Se consideră numerele naturale din mulțimea  $\{0, 1, 2, \dots, n\}$ . Elaborați un program care determină pentru câte numere  $K$  din această mulțime suma cifrelor fiecărui număr este egală cu  $m$ . În particular, pentru  $n = 100$  și  $m = 2$ , în mulțimea  $\{0, 1, 2, \dots, 100\}$  există 3 numere care satisfac condițiile problemei: 2, 11 și 20. Prin urmare,  $K = 3$ .

**Rezolvare.** Evident, mulțimea soluțiilor posibile  $S = \{0, 1, 2, \dots, n\}$ . În programul ce urmează suma cifrelor oricărui număr natural  $i, i \in S$  se calculează cu ajutorul funcției `SumaCifrelor`. Separarea cifrelor zecimale din scrierea numărului natural  $i$  se efectuează de la dreapta la stânga prin împărțirea numărului  $i$  și a cîturilor respective la baza 10.

## Pascal

```
Program P151;
{ Suma cifrelor unui numar natural }
type Natural=0..MaxInt;
var i, K, m, n : Natural;

function SumaCifrelor(i:Natural):Natural;
var suma : Natural;
begin
  suma:=0;
  repeat
    suma:=suma+(i mod 10);
    i:=i div 10;
  until i=0;
  SumaCifrelor:=suma;
end; { SumaCifrelor }

function SolutiePosibila(i:Natural):boolean;
begin
  if SumaCifrelor(i)=m then SolutiePosibila:=true
  else SolutiePosibila:=false;
end; { SumaCifrelor }

procedure PrelucrareaSolutiei(i:Natural);
begin
  writeln('i=', i);
  K:=K+1;
end; { PrelucrareaSolutiei }

begin
  write('Dati n='); readln(n);
  write('Dati m='); readln(m);
  K:=0;
  for i:=0 to n do
    if SolutiePosibila(i) then PrelucrareaSolutiei(i);
  writeln('K=', K);
  readln;
end.
```

## C++

```
// Program P151
// Suma cifrelor unui numar natural
#include <iostream>
using namespace std;
int i, K = 0, m, n;

int SumaCifrelor(int i)
{
```

```

if (i == 0) return 0;
else return (i % 10) + SumaCifrelor(i / 10);
} // SumaCifrelor

bool SolutiePosibila(int i)
{
    if (SumaCifrelor(i)== m) return true;
    else return false;
} // SolutiePosibila

void PrelucrareaSolutiei(int i)
{
    cout << " i = " << i << endl;
    K++;
} // PrelucrareaSolutiei

int main()
{
    cout << " Dati n= "; cin >> n;
    cout << " Dati m= "; cin >> m;
    for (int i = 1; i <= n; i++)
        if (SolutiePosibila(i)) PrelucrareaSolutiei(i);
    cout << "K = " << K ;
}

```

Din analiza programului P151 rezultă că complexitatea temporară a algoritmului respectiv este  $O(n)$ .

**Exemplul 2.** Se consideră mulțimea  $P = \{P_1, P_2, \dots, P_n\}$  formată din  $n$  puncte ( $2 \leq n \leq 30$ ) pe un plan euclidian. Fiecare punct  $P_j$  este definit prin coordonatele sale  $x_j, y_j$ . Elaborați un program care afișează la ecran coordonatele punctelor  $P_a, P_b$  distanța dintre care este maximă.

**Rezolvare.** Mulțimea soluțiilor posibile  $S = P \times P$ . Elementele  $(P_j, P_m)$  ale produsului cartezian  $P \times P$  pot fi generate cu ajutorul a două cicluri imbricate:

### Pascal

```

for j:=1 to n do
    for m:=1 to n do
        if SolutiePosibila(Pj, Pm) then PrelucrareaSolutiei(Pj, Pm)

```

### C++

```

for (int j = 1; j <= n; j++)
    for (int m = 1; m <= n; m++)
        if (SolutiePosibila(Pj, Pm)) PrelucrareaSolutiei(Pj, Pm)

```

Distanța dintre punctele  $P_j, P_m$  se calculează cu ajutorul formulei:

$$d_{jm} = \sqrt{(x_j - x_m)^2 + (y_j - y_m)^2}$$

## Pascal

```
Program P152;
  { Puncte pe un plan euclidian }
const nmax=30;
type Punct = record
  x, y : real;
end;
  Indice = 1..nmax;
var P : array[Indice] of Punct;
j, m, n : Indice;
dmax : real; { distanta maxima }
PA, PB : Punct;

function Distanta(A, B : Punct) : real;
begin
  Distanta:=sqrt(sqr(A.x-B.x)+sqr(A.y-B.y));
end; { Distanta }

function SolutiePosibila(j,m:Indice):boolean;
begin
  if j<>m then SolutiePosibila:=true
    else SolutiePosibila:=false;
end; { SolutiePosibila }

procedure PrelucrareaSolutiei(A, B : Punct);
begin
  if Distanta(A, B)>dmax then
  begin
    PA:=A; PB:=B;
    dmax:=Distanta(A, B);
  end;
end; { PrelucrareaSolutiei }

begin
  write('Dati n='); readln(n);
  writeln('Dati coordonatele x, y ale punctelor');
  for j:=1 to n do
  begin
    write('P[ ', j, ']: '); readln(P[j].x, P[j].y);
  end;
  dmax:=0;

  for j:=1 to n do
    for m:=1 to n do
      if SolutiePosibila(j, m) then
        PrelucrareaSolutiei(P[j], P[m]);

  writeln('Solutia: PA=(', PA.x:5:2, ' ', PA.y:5:2, ')');
  writeln('          PB=(', PB.x:5:2, ' ', PB.y:5:2, ')');
  readln;
end.
```

**C++**

```
// Program P152
// Puncte pe un plan euclidian
#include <iostream>
#include <math.h>
using namespace std;
#define nmax 30

struct Punct {double x; double y;} P[30], PA, PB;
int m, n;
double dmax = 0; // distanta maxima

double Distanta(struct Punct A, struct Punct B)
{
    return sqrt(pow((A.x-B.x),2)+ pow((A.y-B.y),2));
} // Distanta

bool SolutiePosibila(int j, int m)
{
    if (j != m) return true;
    else return false;
} // SolutiePosibila

void PrelucrareaSolutiei(struct Punct A, struct Punct B)
{
    if (Distanta(A, B) > dmax)
    {
        PA = A; PB = B;
        dmax = Distanta(A, B);
    }
} // PrelucrareaSolutiei

int main()
{
    cout << " Dati n= "; cin >> n ;
    cout << "Dati coordonatele x, y ale punctelor " << endl;
    for (int j = 1; j <= n; j++)
    {
        cout << " P[ " << j << " ]: "; cin >> P[j].x >> P[j].y;
    }

    for (int j = 1; j<= n; j++)
        for (int m = j + 1; m <= n; m++)
            if (SolutiePosibila(j, m)) PrelucrareaSolutiei(P[j], P[m]);
    cout << " Solutia: PA=( " << PA.x << " " << PA.y << ")" << endl;
    cout << " PB=( " << PB.x << " " << PB.y << ")" << endl;
}
```

Întrucât partea executabilă a programului P152 conține două cicluri imbricate, complexitatea lui este  $O(n^2)$ .

Din exemplele prezentate mai sus se observă că în algoritmii bazați pe metoda trierii se calculează, implicit sau explicit, mulțimea soluțiilor posibile  $S$ . În problemele relativ simple (*exemplul 1*) elementele din mulțimea soluțiilor posibile pot fi enumerate direct. În problemele mai complicate (*exemplul 2*) generarea soluțiilor posibile necesită elaborarea unor algoritmi speciali. În general, acești algoritmi realizează operațiile legate de prelucrarea unor mulțimi:

- reuniunea;
- intersecția;
- diferența;
- generarea tuturor submulțimilor;
- generarea elementelor unui produs cartezian;
- generarea permutărilor, aranjamentelor sau a combinărilor de obiecte și.a.

Avantajul principal al algoritmilor bazați pe metoda trierii constă în faptul că programele respective sunt relativ simple, iar depanarea lor nu necesită teste sofisticate. Complexitatea temporală a acestor algoritmi este determinată de numărul de elemente  $k$  din mulțimea soluțiilor posibile  $S$ . În majoritatea problemelor de o reală importanță practică metoda trierii conduce la algoritmii exponențiali. Întrucât algoritmii exponențiali sunt inaceptabili în cazul datelor de intrare foarte mari, metoda trierii este aplicată numai în scopuri didactice sau pentru elaborarea unor programe timpul de execuție al căror nu este critic.

## Întrebări și exerciții

- 1 Sistematizează-ți cunoștințele!** Explicați structura generală a algoritmilor bazați pe metoda trierii.
- 2 Explică!** Cum poate fi realizată trierea soluțiilor posibile cu ajutorul ciclurilor?
- 3 Analizează și aplică!** Estimați timpul de execuție a programelor P151 și P152. Modificați programul P152 în aşa mod, încât timpul de execuție să se micșoreze aproximativ de două ori.
- 4** Dați exemple de programe timpul de execuție al căror nu este critic.
- 5 Analizează!** Care sunt avantajele și dezavantajele algoritmilor bazați pe metoda trierii?
- 6 Programează!** Se consideră mulțimea  $P = \{P_1, P_2, \dots, P_n\}$  formată din  $n$  puncte ( $3 \leq n \leq 30$ ) pe un plan euclidian. Fiecare punct  $P_j$  este definit prin coordonatele sale  $x_j, y_j$ . Elaborați un program ce determină trei puncte din mulțimea  $P$  pentru care aria triunghiului respectiv este maximă. Estimați timpul de execuție a programului elaborat.
- 7 Programează!** Scrieți o funcție care, primind ca parametru un număr natural  $n$ , returnează valoarea `true` dacă  $n$  este prim și `false` în caz contrar. Estimați complexitatea temporală a funcției respective.
- 8 Programează!** În notația  $(a)_x$ , litera  $x$  reprezintă baza sistemului de numerație, iar litera  $a$  – un număr scris în sistemul respectiv. Elaborați un program care calculează, dacă există, cel puțin o rădăcină a ecuației

$$(a)_x = b,$$

unde  $a$  și  $b$  sunt numere naturale, iar  $x$  este necunoscută. Fiecare cifră a numărului natural  $a$  aparține mulțimii  $\{0, 1, 2, \dots, 9\}$ , iar numărul  $b$  este scris în sistemul zecimal. De exemplu, rădăcina ecuației

$$(160)_x = 122$$

este  $x = 8$ , iar ecuația

$$(5)_x = 10$$

nu are soluții. Estimați complexitatea temporală a programului elaborat.

9

**Programează!**

Într-o pușculită se află  $N$  monede de diferite valori cu greutatea totală  $G$  grame. Greutatea fiecărei monede de o anumită valoare este dată în tabelul ce urmează.

Valoarea monedei, lei	Greutatea monedei, grame
1	4,45
2	6,7
5	7,1
10	7,65

Elaborați un program care determină:

- a) suma minimă  $S_{min}$  care ar putea să fie în pușculită;
- b) suma maximă  $S_{max}$  care ar putea să fie în pușculită.

Estimați complexitatea temporală a programului elaborat.

10

**Programează!**

Elaborați un program care determină câte puncte cu coordonate întregi se conțin într-o sferă de raza  $R$  cu centrul în originea sistemului de coordonate. Se consideră că  $R$  este un număr natural,  $1 \leq R \leq 30$ . Distanța  $d$  dintre un punct cu coordonatele  $(x, y, z)$  și originea sistemului de coordonate se determină după formula  $d = \sqrt{x^2 + y^2 + z^2}$ . Estimați complexitatea temporală a programului elaborat.

## 2.8

**Tehnica Greedy**

În această metodă se presupune că problemele pe care trebuie să le rezolvăm au următoarea structură:

- se dă o mulțime  $A = \{a_1, a_2, \dots, a_n\}$  formată din  $n$  elemente;
- se cere să determinăm o submulțime  $B$ ,  $B \subseteq A$ , care îndeplinește anumite condiții pentru a fi acceptată ca soluție.

În principiu, problemele de acest tip pot fi rezolvate prin metoda trierii, generând consecutiv cele  $2^n$  submulțimi  $A_i$  ale mulțimii  $A$ . Dezavantajul metodei trierii constă în faptul că timpul cerut de algoritmii respectivi este foarte mare.

Pentru a evita trierea tuturor submulțimilor  $A_i$ ,  $A_i \subseteq A$ , în metoda *Greedy* se utilizează un **criteriu (o regulă)** care asigură alegerea directă a elementelor necesare din mulțimea  $A$ . De obicei, criteriile sau regulile de selecție nu sunt indicate explicit în enunțul problemei și formularea lor cade în sarcina programatorului. Evident, în absența unor astfel de criterii metoda *Greedy* nu poate fi aplicată.

**Schema generală** a unui algoritm bazat pe metoda *Greedy* poate fi redată cu ajutorul unui ciclu:

### Pascal

```
while ExistaaElemente do
begin
    AlegeUnElement(x);
    IncludeElementul(x);
end
```

### C++

```
while (ExistaaElemente)
{
    AlegeUnElement(x);
    IncludeElementul(x);
}
```

Funcția `ExistaaElemente` returnează valoarea `true` dacă în mulțimea  $A$  există elemente care satisfac criteriul (regula) de selecție. Procedura `AlegeUnElement` extrage din mulțimea  $A$  un astfel de element  $x$ , iar procedura `IncludeElementul` include elementul selectat în submulțimea  $B$ . Inițial  $B$  este o mulțime vidă.

După cum se vede, în metoda *Greedy* soluția problemei se caută prin testarea consecutivă a elementelor din mulțimea  $A$  și prin includerea unora dintre ele în submulțimea  $B$ . Într-un limbaj plastic, submulțimea  $B$  încearcă să „îngheță” elementele „gustoase” din mulțimea  $A$ , de unde provine și denumirea metodei (*greedy* – lacom, hrăpăret).

*Exemplu.* Se consideră mulțimea  $A=\{a_1, a_2, \dots, a_i, \dots, a_n\}$  elementele căreia sunt numere reale, iar cel puțin unul dintre ele satisface condiția  $a_i > 0$ . Elaborați un program care determină o submulțime  $B$ ,  $B \subseteq A$ , astfel încât suma elementelor din  $B$  să fie maximă. De exemplu, pentru  $A=\{21,5; -3,4; 0; -12,3; 83,6\}$  avem  $B=\{21,5; 83,6\}$ .

*Rezolvare.* Se observă că dacă o submulțime  $B$ ,  $B \subseteq A$  conține un element  $b \leq 0$ , atunci suma elementelor submulțimii  $B \setminus \{b\}$  este mai mare sau egală cu cea a elementelor din  $B$ . Prin urmare, regula de selecție este foarte simplă: la fiecare pas, în submulțimea  $B$  se include un element pozitiv arbitrar din mulțimea  $A$ .

În programul ce urmează mulțimile  $A$  și  $B$  sunt reprezentate prin vectorii (tablourile unidimensionale)  $A$  și  $B$ , iar numărul de elemente ale fiecărei mulțimi – prin variabilele întregi  $n$  și, respectiv,  $m$ . Inițial  $B$  este o mulțime vidă și, respectiv,  $m=0$ .

### Pascal

```
Program P153;
{ Tehnica Greedy }
const nmax=1000;
var A : array [1..nmax] of real;
    n : 1..nmax;
    B : array [1..nmax] of real;
    m : 0..nmax;
```

```

x : real;
i : 1..nmax;

Function ExistaElemente : boolean;
var i : integer;
begin
  ExistaElemente:=false;
  for i:=1 to n do
    if A[i]>0 then ExistaElemente:=true;
  end; { ExistaElemente }

procedure AlegeUnElement(var x : real);
var i : integer;
begin
  i:=1;
  while A[i]<=0 do i:=i+1;
  x:=A[i];
  A[i]:=0;
end; { AlegeUnElement }

procedure IncludeElementul(x : real);
begin
  m:=m+1;
  B[m]:=x;
end; { IncludeElementul }

begin
  write('Dati n='); readln(n);
  writeln('Dati elementele multimii A:');
  for i:=1 to n do read(A[i]);
  writeln;
  m:=0;
  while ExistaElemente do
    begin
      AlegeUnElement(x);
      IncludeElementul(x);
    end;
  writeln('Elementele multimii B:');
  for i:=1 to m do writeln(B[i]);
  readln;
end.

```

Mentionăm că procedura `AlegeUnElement` zerografiază componenta vectorului A ce conține elementul  $x$ , extras din mulțimea respectivă.

## C++

```
// Program P153
// Tehnica Greedy
#include <iostream>
using namespace std;
#define nmax 1000

double A[nmax], B[nmax], k;
int n, m = 0, d;

int AlegeElementExistent()
{
    for (int i = 1; i<= n; i++)
        if (A[i] > 0) return i;
    return -1;
} // AlegeElementExistent

void IncludeElementulAles(int k)
{
    m++;
    B[m] = A[k];
    A[k] = 0;
} // IncludeElementulAles

int main()
{
    cout <<" Dati n= "; cin >> n ;
    cout << "Dati elementele multimii A " << endl;
    for (int j = 1; j <= n; j++) cin >> A[j];
    do
    {
        d = AlegeElementExistent();
        if (d > 0) IncludeElementulAles (d);
    } while (d > 0);
    cout << " Elementele multimii B: " << endl;
    for (int i = 1; i <= m; i++) cout << B[i] << " ";
}
```

În varianta C/C++ a programului P153 funcția `AlegeElementExistent` combină efectul funcțiilor `ExistăElemente` și `AlegeUnElement`, iar zerografierea componentelor vectorului `A`, transferate în mulțimea `B`, este realizată în funcția `IncludeElementulAles`.

**Complexitatea temporală** a algoritmilor bazați pe metoda *Greedy* poate fi evaluată urmând schema generală de calcul prezentată mai sus. De obicei, timpul cerut de subprogramele `ExistăElemente`, `AlegeUnElement` și `IncludeElementul` este de ordinul  $n$ . În ceea ce privește ciclul `while` aceste proceduri se execută cel mult de  $n$  ori. Prin urmare, algoritmii bazați pe metoda *Greedy* sunt algoritmi polinomiali. Pentru comparare, amintim că algoritmii bazați pe trierea tuturor submulțimilor  $A_i$ ,  $A_i \subseteq A$  sunt algoritmi de ordinul  $O(2^n)$ , deci exponențiali. Cu regret, metoda *Greedy* poate fi aplicată numai atunci când din enunțul problemei poate fi deducătoare care asigură selecția directă a elementelor necesare din mulțimea  $A$ .

Accentuăm faptul că metoda *Greedy* nu garantează găsirea unei soluții optime, iar exactitatea acesteia depinde de criteriul (regula) de selecție, implementată de programator.

## Întrebări și exerciții

- 1 Sistematizează-ți cunoștințele!** Explicați structura generală a algoritmilor bazați pe metoda Greedy.
- 2 Analizează!** Care sunt avantajele și neajunsurile algoritmilor bazați pe tehnica Greedy?
- 3 Exersează!** Estimați timpul de execuție a programului P153.
- 4 Studiu de caz** Schițați un algoritm care determină submulțimea  $B$  din programul P153 prin metoda trierii. Estimați complexitatea temporală a algoritmului elaborat.
- 5 Integrează cunoștințele și aplică!** **Memorarea fișierelor pe benzi magnetice.** Se consideră  $n$  fișiere  $f_1, f_2, \dots, f_n$  care trebuie memorate pe o bandă magnetică. Elaborați un program care determină ordinea de amplasare a fișierelor pe bandă astfel, încât timpul mediu de acces să fie minim. Se presupune că frecvența de citire a fișierelor este aceeași, iar pentru citirea fișierului  $f_i$  ( $i=1, 2, \dots, n$ ) sunt necesare  $t_i$  secunde.
- 6 Integrează cunoștințele și aplică!** **Problema continuă a rucsacului.** Se consideră  $n$  obiecte. Pentru fiecare obiect  $i$  ( $i=1, 2, \dots, n$ ) se cunoaște greutatea  $g_i$  și câștigul  $c_i$  care se obține în urma transportului său la destinație. O persoană are un rucsac cu care pot fi transportate unul sau mai multe obiecte greutatea sumară a cărora nu depășește valoarea  $G_{max}$ . Elaborați un program care determină ce obiecte trebuie să transporte persoana în aşa fel încât câștigul să fie maxim. În caz de necesitate unele obiecte pot fi tăiate în fragmente mai mici.
- 7 Integrează cunoștințele și aplică!** **Hrubele de la Cricova.** După o vizită la renumitele hrube\* de la Cricova, un informatician a construit un robot care funcționează într-un câmp divizat în pătrățele (fig. 2.5). Robotul poate executa doar instrucțiunile SUS, JOS, DREAPTA, STANGA, conform cărora se deplasează în unul dintre pătrățele vecine. Dacă în acest pătrat este un obstacol, de exemplu, un perete sau un butoi, are loc un accident și robotuliese din funcție.

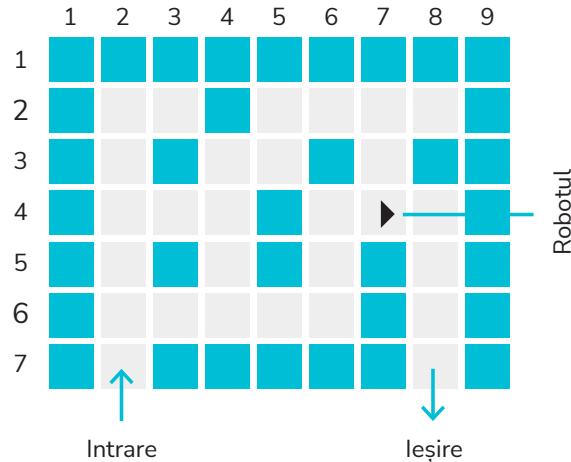


Fig. 2.5. Câmpul de acțiune al robotului

\* Hrubă – încăpere sau galerie subterană care servește la depozitarea produselor alimentare. În hrubele de la Cricova pe parcursul a mai multor decenii au fost depozitate cele mai bune vinuri din Republica Moldova.

Elaborați un program care, cunoscând planul hrubelor deplasează robotul prin încăperile subterane, de la intrarea în hrube la ieșire. Colectia de vinuri fiind foarte bogată, nu se cere vizitarea obligatorie a tuturor încăperilor subterane.

*Datele de intrare.* Planul hrubelor este desenat pe o foaie de hârtie liniată în pătrățele. Pătrățelele hașurate reprezintă obstacolele, iar cele nehașurate – spațiile libere. Pătrățelele de pe perimetru planului, cu excepția celor de intrare sau ieșire, sunt hașurate prin definiție. În formă numerică, planul hrubelor este redat prin tabloul A cu  $m$  linii și  $n$  coloane. Elementele  $A[i, j]$  ale acestui tablou au următoarea semnificație: 0 – spațiu liber; 1 – obstacol; 2 – intrarea în hrube; 3 – ieșirea din hrube. Inițial, robotul se află în pătrățelul pentru care  $A[i, j]=2$ .

*Fișierul HRUBE . IN* conține pe prima linie numerele  $m, n$  separate prin spațiu. Pe următoarele  $m$  linii se conțin câte  $n$  numere  $A[i, j]$  separate prin spațiu.

*Datele de ieșire.* Fișierul HRUBE . OUT va conține pe fiecare linie câte una dintre instrucțiunile SUS, JOS, DREAPTA, STANGA scrise în ordinea executării lor de către robot.

*Restrictii.*  $5 \leq m, n \leq 100$ . Timpul de execuție nu va depăși 3 secunde.

**Exemplu:**

HRUBE . IN

7	9							
1	1	1	1	1	1	1	1	1
1	0	0	1	0	0	0	0	1
1	0	1	0	0	1	0	1	1
1	0	0	0	1	0	0	0	1
1	0	1	0	1	0	1	0	1
1	0	0	0	0	0	1	0	1
1	2	1	1	1	1	1	3	1

HRUBE . OUT

SUS
DREAPTA
DREAPTA
DREAPTA
DREAPTA
SUS
SUS
DREAPTA
DREAPTA
JOS
JOS
JOS

*Indicații.* La fiecare pas selectați din multimea {SUS, JOS, DREAPTA, STANGA} câte o instrucțiune în aşa fel încât robotul să se depleteze numai de-a lungul unui perete. Evident, acest criteriu (regulă) de selecție nu garantează găsirea celui mai scurt drum. Întrucât problema admite mai multe soluții, programul va calcula doar una dintre ele.

# Modelare și calcul numeric

## 3.1 Noțiune de model. Clasificarea modelelor

Imitarea unor procese, obiecte sau fenomene este caracteristică societății umane pe tot parcursul istoriei sale. Primele desene, realizate de oamenii epocii de piatră pe pereții peșterilor, erau în același timp și primele încercări de a reproduce obiectele și fenomenele reale prin imagini (fig. 3.1).

Globul-machetă al planetei noastre este și el o imitație a unui corp real. El ne aduce la cunoștință date despre forma și mișcarea Terrei, amplasarea continentelor și oceanelor, a țărilor și orașelor (fig. 3.2). Dar elementele machetei nu constituie în întregime obiectul inițial. Astfel, în cazul globului-machetă avem de-a face doar cu un corp sferic, prin centrul căruia trece o axă care permite rotirea, iar pe suprafață având imprimate diverse informații despre planeta Pământ. Globul-machetă redă anumite trăsături ale corpului cosmic real, dar se deosebește de el: diferă dimensiunile, proprietățile fizice, structura etc. Globul-machetă este o reprezentare simplificată a Terrei, care permite studierea doar a anumitor particularități ale ei – este doar un *model*.

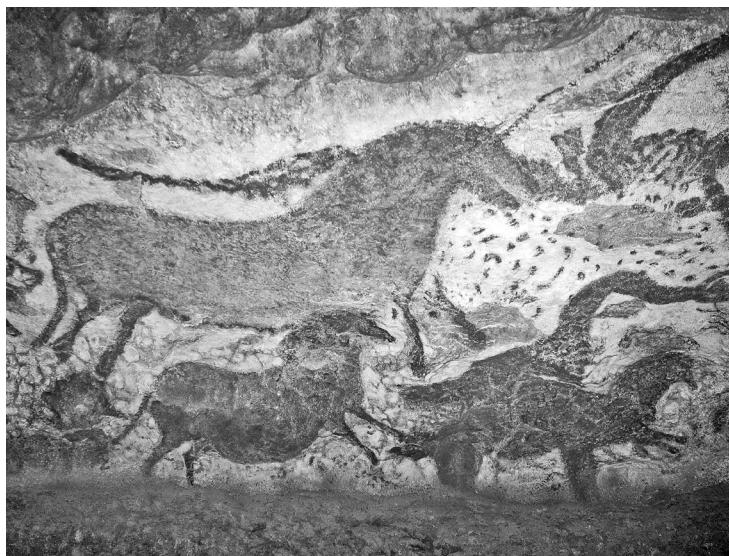


Fig. 3.1. Imagine zoomorfă din paleolitic



Fig. 3.2. Globul-machetă al Pământului

**Modelul este un sistem material sau ideal, logico-matematic cu ajutorul căruia pot fi studiate, prin analogie, proprietățile și operațiile efectuate asupra sistemului inițial, care, în general, este mai complex.**

Modelele sunt utilizate din cele mai vechi timpuri pentru studierea fenomenelor și proceselor complexe, de exemplu, a moleculelor și a atomilor, a Sistemului Solar și a universului în ansamblu, a reactoarelor atomice, a zgârie-norilor etc. Un model reușit este mai comod pentru cercetări decât obiectul real. Mai mult chiar, anumite obiecte și fenomene nici nu pot fi studiate în original. De exemplu, sunt greu de efectuat experimente ce țin de economia unei țări, sunt imposibile experiențele cu planetele Sistemului Solar, cele care presupun revenirea în timp etc. Un alt aspect important al modelării îl constituie posibilitatea de a pune în evidență doar acei factori, acele proprietăți ale obiectului real care sunt esențiale pentru obiectul studiat.

De asemenea, modelul permite instruirea în vederea utilizării corecte a obiectului real, verificând diferite moduri de a reacționa pe modelul acestui obiect. Experiențele cu obiectul real pot fi imposibile sau foarte periculoase (durata mare a procesului în timp, riscul de a deteriora obiectul). În cazurile cercetării obiectelor dinamice, caracteristicile cărora depind de timp, o importanță primordială capătă problema prognozării stării obiectului sub acțiunea unor anumiți factori.

În general, un model bine construit permite obținerea unor cunoștințe noi despre obiectul original supus cercetării.

### I Procesul de construire a modelului se numește modelare.

Există câteva tipuri de modelare, ce pot fi unite în două grupe mari: *modelarea materială* și *modelarea ideală*.

În cazul **modelării materiale**, cercetarea originalului se efectuează prin redarea cu ajutorul unui alt obiect material, mai simplu decât cel real, denumit model, a caracteristicilor geometrice, fizice, dinamice, funcționale de bază ale originalului. Exemple: machetele clădirilor, avioanelor, automobilelor și ale vehiculelor militare etc.

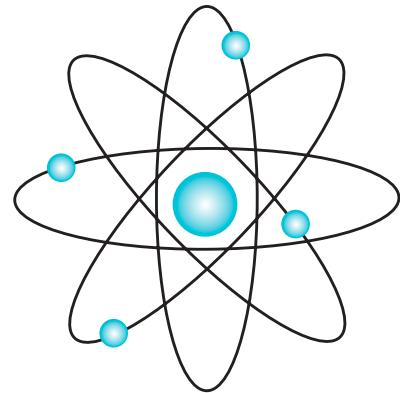
În cazul **modelării ideale**, cercetarea originalului se efectuează prin reprezentarea proprietăților lui cu ajutorul unor concepte, scheme, planuri, structuri, care există doar în imaginea omului.

În general, *modelarea ideală* se bazează pe o concepție intuitivă despre obiectul cercetărilor. De exemplu, experiența de viață a fiecărui om poate fi considerată drept un model personificat al lumii înconjurătoare. Atunci când modelarea nu este intuitivă și se folosesc anumite simboluri, cum ar fi diverse scheme, grafice, formule, ea este numită *simbolică*. În categoria modelelor simbolice un loc aparte îl ocupă *modelarea matematică*, în care examinarea obiectului se realizează prin intermediul unui model formulat în termeni și notiuni matematice, cu folosirea unor metode matematice. Un exemplu clasic al modelării matematice îl constituie descrierea și cercetarea legilor de bază ale mecanicii lui Newton cu ajutorul instrumentelor matematice.

Pentru studierea unui proces sau fenomen nu este suficient să fie construit modelul respectiv. În general, modelul descrie anumite legități, relații, caracteristici ale originalului, însă scopul modelării constă în obținerea, în baza informațiilor furnizate de model, a unor date noi despre original.

## Întrebări și exerciții

- 1 Sistematizează-ți cunoștințele!** Explicați noțiunea de model. Dați exemple de obiecte ale lumii înconjurătoare și modele ale acestora. Pentru fiecare exemplu, indicați caracteristicile obiectului original ce sunt redate de model.
- 2 Explică!** Formulați definiția noțiunii de model material. Dați exemple de modele materiale. Argumentați necesitatea utilizării modelării materiale în diferite domenii ale științei și tehnicii.
- 3 Explică!** Formulați definiția noțiunii de model ideal. Exemplificați. Motivați necesitatea utilizării modelelor理想的.
- 4 Integrează!** În anul 1911, cercetătorul E. Rutherford a propus „modelul planetar (nuclear) al atomului”. Explicați sensul cuvântului „model” în contextul propunerii lui Rutherford, precum și motivul pentru care el este numit planetar.
- 5 Explicați sensul următoarei afirmații: „Pictogramele interfeței grafice a utilizatorului reprezintă modele ale obiectelor prelucrate de produsele software”.



### 3.2 Modelul matematic și modelarea matematică

Unul dintre scopurile de bază ale informaticii, ca știință interdisciplinară, constă în elaborarea metodelor de rezolvare a problemelor complicate de cercetare și de calcul cu ajutorul tehnicii computaționale. Inițial, informatica se dezvoltă ca o ramură a matematicii aplicate. Primele probleme, abordate în cadrul informaticii, erau, de asemenea, pur matematice, iar soluționarea lor se reducea la efectuarea unui volum mare de calcule complexe. În prezent însă, informatica a devenit o știință independentă, cu propriile metode și obiecte de cercetare, care se bazează pe legitățile matematice. Informatica studiază și soluționează probleme complexe din domeniul matematicii, fizicii, chimiei, biologiei, economiei, ecologiei, filologiei și al sociologiei. Dar, indiferent de domeniul din care provine problema, în procesul de soluționare a ei informatica se bazează pe matematică. În consecință, pentru a dezlega o anumită problemă, înainte de a utiliza calculatorul propriu-zis, este necesară descrierea fenomenelor și proceselor care apar în problema supusă rezolvării cu ajutorul noțiunilor matematice. În particular, acestea pot fi funcții, ecuații, inecuații, sisteme de ecuații etc.

**Modelul matematic reprezintă descrierea unui proces sau a unui fenomen cu ajutorul noțiunilor matematice.**

**Exemplul 1:** Se consideră două automobile. Unul dintre ele se mișcă rectiliniu, traiectoria respectivă fiind definită prin ecuația  $x = 1/2$ . Al doilea automobil se deplasează pe o traiectorie circulară, descrisă de o circumferință unitară cu centrul în originea sistemului de coordinate (fig. 3.3). Se cere determinarea coordonatelor punctelor posibile de impact ale automobilelor.

Abstractizând problema, obținem următoarea formulare: un punct se mișcă pe o traiectorie dată de ecuația  $x = 1/2$ . Al doilea punct se mișcă pe o traiectorie dată de ecuația  $x^2 + y^2 = 1$ . Se cere să se găsească soluțiile sistemului de ecuații:

$$\begin{cases} x = \frac{1}{2} \\ x^2 + y^2 = 1 \end{cases}$$

Grafic problema este reprezentată de schema alăturată.

Algoritmul de rezolvare e următorul:

1. Se consideră  $x = 1/2$ .
2. Din ecuația a doua a sistemului se calculează

$$y_1 = \sqrt{1 - \frac{1}{4}}; \quad y_2 = -\sqrt{1 - \frac{1}{4}}$$

3. Soluția problemei este  $\left(\frac{1}{2}, \frac{\sqrt{3}}{2}\right)$ ;  $\left(\frac{1}{2}, -\frac{\sqrt{3}}{2}\right)$ .

**Exemplul 2:** Pe o masă netedă se află o bilă metalică, fixată de un arc (fig. 3.4). Arcul se comprimă, fără a-i deteriora elasticitatea, apoi se eliberează. Se cere determinarea coordonatei bilei peste  $t$  secunde.

Dacă  $k$  – coeficientul de elasticitate a arcului,  $m$  – masa bilei,  $x$  – mărimea deformației arcului, atunci, în baza legii lui Hooke și a legii 2 a lui Newton, modelul matematic al sistemului bilă–arc va avea forma:

$$ma = -kx, \text{ unde } a \text{ – accelerația.}$$

Poziția bilei după comprimarea arcului (deformația inițială) se notează prin  $x_0$  (fig. 3.5). Se cere determinarea poziției bilei (mărimea deformației) peste  $t$  secunde. Pentru aceasta, modelul precedent se va transforma astfel încât să fie prezentată dependența valorii  $x$  (a deformației) de timp. Pentru deformări mici și lipsa forței de frecare se va folosi legea deplasărilor armonice

$$x(t) = x_0 \cos \sqrt{\frac{k}{m}} t.$$

Formula dată permite determinarea poziției bilei  $x$  (deformația arcului) în orice moment de timp  $t$ , în situația în care sunt cunoscute valorile  $k$ ,  $m$  și  $x_0$ .

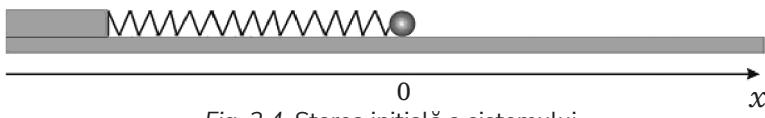


Fig. 3.4. Starea inițială a sistemului



Fig. 3.5. Starea sistemului după comprimarea arcului

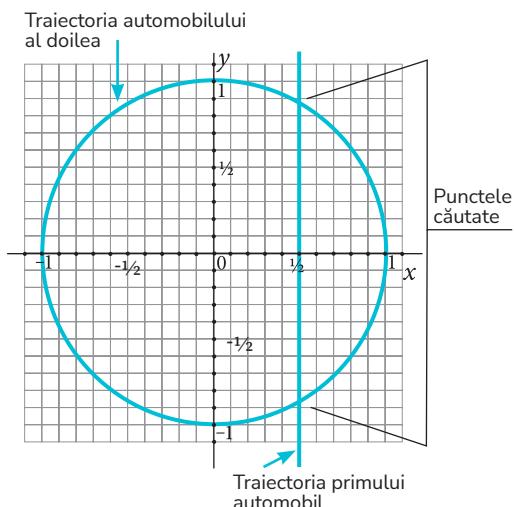


Fig. 3.3. Deplasarea automobilelor

Folosind formula dedusă mai sus, se poate rezolva problema generală de determinare a deformației în orice moment de timp cu ajutorul următorului program:

```
program cn01;
var
t, x0,k,m,x: real;
begin
readln(x0,k,m,t);
x:=x0*cos(sqrt(k/m*t));
writeln('x=', x:0:6);
end.
```

```
// program cn01;
#include <iostream>
#include <math.h>
using namespace std;
double t, x0,k,m,x;
int main()
{
    cin >> x0 >> k >> m >>t;
    x =x0 * cos(sqrt(k / m * t));
    cout << "x = " << x;
    return 0;
}
```

## Întrebări și exerciții

- 1 Sistematizează-ți cunoștințele!** Definiți noțiunea de model matematic. Este el oare un model material?
- 2 Integrează!** Indicați câteva domenii de utilizare a modelelor matematice. Argumentați necesitatea întrebuișterii lor.
- 3 Argumentează!** Motivați necesitatea transpunerii modelelor matematice în programe de calculator și a folosirii acestora din urmă.
- 4 Explică!** Elaborați un program care, pentru problema din exemplul 2, va calcula deformația arcului  $x$  cu intervale de o secundă, din momentul inițial până în momentul  $t$  ( $t$  are o valoare întreagă). Valorile  $x_0, k, m, t$  se vor introduce de la tastatură.
- 5 Explică!** Izotopul radioactiv plutoniu-235 are perioada de înjumătățire de 26 de minute. În această perioadă, jumătate din cantitatea inițială a izotopului dispare prin descompunere în alte elemente chimice. Descrieți modelul matematic care permite calculul numărului ciclurilor de înjumătățire necesare pentru dispariția a  $k$  procente din cantitatea inițială a izotopului.
- 6 Aplică!** Efectul unui medicament se calculează conform formulei  $r_k = ar_{k-1} + 0,4^k$ , unde  $r_k$  este concentrația substanțelor active peste  $k$  ore după administrarea lui. Inițial  $r_0 = 1$ , iar  $0 < a < 1$ . Din formulă rezultă că  $r_k$  atinge valoarea maximă  $r_m$  după ce au trecut  $m$  ore, ulterior începând să scadă. Scrieți un program ce va determina peste câte ore efectul medicamentului în studiu va atinge valoarea maximă. Numărul real  $a$  se citește de la tastatură.

### 3.3 Soluții analitice și soluții de simulare

Mai întâi vom examina următoarea problemă:

**Exemplul 1:** Un bazin cu volumul  $100 \text{ m}^3$ , care inițial conține  $20 \text{ m}^3$  de apă, se umple cu același lichid folosind pompa ce are capacitatea de pompare de  $15 \text{ m}^3/\text{oră}$ . Totodată, din bazin în dispozitivul de filtrare se scurg  $5 \text{ m}^3/\text{oră}$ . Se cere să se determine peste câte ore va fi umplut bazinul.

Desigur, există mai multe metode de rezolvare a acestei probleme. Una dintre cele mai simple este modelarea procesului de acumulare a apei în bazin peste fiecare oră, cu ajutorul unui tabel.



Timp (ore)	Cantitatea de apă ce a fost pompată	Cantitatea de apă ce s-a scurs	Cantitatea de apă din bazin
0	0	0	20
1	15	5	30
2	30	10	40
3	45	15	50
4	60	20	60
5	75	25	70
6	90	30	80
7	105	35	90
8	120	40	100

Soluția problemei a fost obținută printr-un număr relativ mare de calcule consecutive, care au reconstruit dinamic volumul de apă în bazin peste fiecare oră. Sigur, nu este cea mai eficientă metodă: se utilizează un număr considerabil de rezultate intermedii, numărul de operații realizate este la fel nemotivat de mare.

O altă soluție se bazează pe o formulă care permite calculul direct al rezultatului final. Timpul necesar pentru umplerea bazinului este determinat de diferența dintre volumul total al bazinului și cantitatea de apă care se conținea inițial în el raportată la debitul apei timp de o oră:

$$t = \frac{100 - 20}{15 - 5} = 8.$$

Prima dintre metodele efectuate utilizează un proces iterativ, care determină volumul de apă după fiecare interval elementar de timp (1 oră) și calculează rezultatele noi, folosind datele obținute la etapa precedentă. Cu alte cuvinte, a fost realizat procesul de modelare a etapelor de umplere a bazinului. Acest proces se numește *simulare*.

**Simularea este o tehnică de rezolvare a problemelor, bazată pe utilizarea unor modele matematice și logice ce descriu comportarea unui sistem real în spațiu și/sau în timp.**

**Model de simulare este modelul de rezolvare a problemei în baza tehnicii de simulare. Soluțiile obținute prin procesul de simulare se numesc soluții de simulare.**

De obicei, modelele de simulare se folosesc atunci când este necesară determinarea stării sistemului cercetat atât în momentul când se obține soluția finală, cât și în momentele intermediare de timp.

Cea de a doua metodă din exemplul de mai sus a realizat calculul direct al soluției prin utilizarea unei formule:

$$t = \frac{V_{bazin} - V_{initial}}{C_{pompă} - C_{scurgere}}, \text{ unde } \begin{aligned} V_{bazin} &= \text{volumul total al bazinului;} \\ V_{initial} &= \text{cantitatea inițială de apă în bazin;} \\ C_{pompă} &= \text{capacitatea de pompare a pompei;} \\ C_{scurgere} &= \text{capacitatea găurii de scurgere.} \end{aligned}$$

Formula permite calculul timpului necesar pentru umplerea *oricărui bazin*, cu *orice cantitate inițială* de apă și *cu orice capacitate de pompare a pompei*, depășind *capacitatea de scurgere*, care de asemenea poate varia.

**Metoda de rezolvare a problemelor bazată pe utilizarea unor formule ce permit calculul direct al rezultatului final, fără a cerceta stări și rezultate intermediare, se numește metodă analitică. Soluțiile obținute cu ajutorul acestei metode sunt numite soluții analitice.**

**Exemplul 2:** Să se scrie un program care calculează suma primilor  $n$  termeni ai progresiei geometrice, având primul termen cu valoarea  $a_1$ , ( $a_1 > 0$ ) și rația  $q$ , ( $q < 1$ ).

Din matematică este cunoscută formula  $S_n = \frac{a_1(q^n - 1)}{q - 1}$ , care permite determinarea directă a sumei, fără a calcula valoarea fiecărui termen al progresiei. Totodată, se știe că termenul cu indicele  $n$ , ( $n \geq 2$ ) poate fi calculat folosind formula recurrentă  $a_n = q \times a_{n-1}$ . Atunci rezultatul poate fi calculat iterativ, prin calculul recurrent al termenilor progresiei și adunarea lor consecutivă  $S_n = a_1 + a_2 + \dots + a_{n-1} + a_n$ .

## Pascal

Calcul direct al sumei	Calcul iterativ al sumei
<pre>program cn02; var S,a,q : real; n : integer; begin   readln(a, q, n);   S:=a*(exp(n*ln(q))-1)/(q-1);   writeln('S=', S:0:6); end.</pre>	<pre>program cn03; var S,a,q : real; i, n : integer; begin   readln(a, q, n);   S:=0;   for i:=1 to n do     begin       S:=S+a;       a:=a*q;     end;   writeln('S=', S:0:6); end.</pre>

```
// program cn02
#include <iostream>
#include <math.h>
using namespace std;

double S, a, q;
int n;

int main()
{
    cin >> a >> q >> n;
    S = a * (exp(n*log(q)) - 1)/
(q - 1);
    cout << "S = " << S;
    return 0;
}
```

```
// program cn03;
#include <iostream>
using namespace std;

double S, a, q;
int n;

int main()
{
    cin >> a >> q >> n;
    S = 0;
    for (int i = 1; i <= n; i++)
    {
        S = S + a;
        a = a * q;
    }
    cout << "S = " << S;
    return 0;
}
```

Fiecare dintre metodele prezentate mai sus are avantajele și neajunsurile sale. Pentru unele probleme este foarte complicat sau practic imposibil de determinat formula analitică (de exemplu, coordonatele unei comete sau ale unui asteroid în funcție de timp), pentru altele este destul de dificil de simulaț un model adecvat, chiar și folosind un număr foarte mare de calcule intermediiare.

Soluția analitică permite calculul imediat al rezultatului final, dar nu permite cercetarea dinamicii construirii acestuia. Utilizarea unui proces iterativ (a soluției de simulare) permite construirea dinamică a rezultatului în funcție de datele folosite în problemă și controlul soluției la fiecare iterată realizată. În același timp, pentru obținerea soluțiilor de simulare, este necesar un număr mai mare de operații decât în cazul soluțiilor analitice.

Alegerea modului de determinare a soluției este influențată de mai mulți factori, principaliii dintre ei fiind:

- posibilitatea de determinare a soluției analitice;
- necesitatea cercetării soluțiilor (stărilor) intermediiare;
- timpul necesar pentru realizarea calculelor (în cazul soluțiilor de simulare);
- eroarea soluției de simulare (diferența dintre mărimea calculată și cea exactă).

## Întrebări și exerciții

- Explică!** Definiți noțiunea de simulare. Ce este o soluție de simulare?
- Sistematizează-ți cunoștințele!** Ce înțelegeți prin metoda analitică de rezolvare a unei probleme? Ce reprezintă o soluție analitică? Care sunt proprietățile soluțiilor analitice?
- Analyzează!** Enumerați proprietățile soluțiilor de simulare. Care dintre aceste proprietăți implică utilizarea calculatorului pentru a găsi astfel de soluții?

**4 Explică!** Elaborați o metodă de simulare pentru calcularea elementului cu numărul  $n$  din sirul de numere 1, 2, 3, 5, 8, 13, 21, ... . Există oare o metodă analitică pentru determinarea elementului cu numărul  $n$ ?

**5 Integrează cunoștințele și aplică!** Rezolvați problemele ce urmează prin metoda simulării:

- În timpul zilei o buburuză urcă pe un stâlp 5 m, iar în timpul noptii coboară 3 m. Ascensiunea începe dimineața. Înălțimea stâlpului este de 15 m. Peste câte zile va ajunge buburuza în vârful stâlpului?
- În condițiile punctului precedent, ascensiunea începe dimineața de la înălțimea de 6 m.
- În condițiile punctului precedent, ascensiunea începe odată cu căderea noptii.

**6 Programează!** Elaborați un program care calculează suma primilor  $n$  termeni ai progresiei aritmetice, având primul termen cu valoarea  $a_1$ , ( $a_1 > 0$ ) și rația  $r$ , ( $r > 0$ ).

### 3.4 Etapele rezolvării problemei la calculator

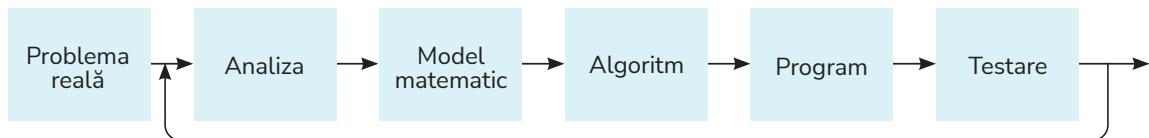
Instrumentele informatici permit rezolvarea problemelor atât prin metode analitice, cât și prin metode de simulare. Indiferent de metoda aplicată, rezolvarea oricărei probleme include mai multe etape, fiecare dintre ele având același grad de importanță.

**Analiza problemei.** Este etapa de studiu al conținutului problemei. Se stabilește setul de date inițiale, se determină care este rezultatul ce urmează să fie obținut, care sunt relațiile dintre datele inițiale și rezultat. Tot la această etapă sunt stabilite restricțiile suplimentare asupra datelor inițiale și a rezultatului.

**Elaborarea modelului matematic al problemei.** La această etapă datele inițiale sunt descrise prin structuri matematice. Folosind limbajul matematic, se descriu relațiile care permit obținerea rezultatului din datele inițiale. În funcție de problemă, aceste relații pot fi recurente (este creat un model de simulare) sau să permită calculul direct al rezultatului (model analitic). Tot aici are loc (dacă este necesar) divizarea problemei în *subprobleme* și elaborarea separată a modelelor matematice pentru fiecare dintre ele.

**Elaborarea algoritmului.** În cazul rezolvării informatici a unei probleme, algoritmul conține setul de instrucțiuni necesare pentru soluționarea problemei, descrise într-o formă pre-stabilită (pseudocod, schemă logică etc.), precum și ordinea executării acestora (pașii algoritmului). Dacă problema a fost divizată în subprobleme, algoritmul, suplimentar la descrierea subalgoritmilor, stabilește modul și condițiile de apel al acestora.

**Scrierea programului.** Pentru rezolvarea automatizată a problemei, cu ajutorul calculatorului, algoritmul trebuie transpus într-o formă înțeleasă de calculator – **program**, folosind un **limbaj de programare**. Pașii algoritmului sunt prezentați cu ajutorul instrucțiunilor limbajului de programare, iar ordinea executării lor – de consecutivitatea și structura instrucțiunilor limbajului. Datele inițiale și intermediare sunt descrise folosind structurile de date, acceptate de limbajul de programare. În procesul de scriere a programului pot să apară erori sintactice și/sau semantice. Procesul de corectare a lor este, de asemenea, o parte a etapei de scriere a programului. Etapa se consideră încheiată atunci când compilarea sau interpretarea programului finalizează fără erori.



**Testarea programului.** O compilare reușită nu garantează rezolvarea corectă a problemei. Pentru verificarea corectitudinii programului, se execută o serie de teste care stabilesc corectitudinea rezultatelor generate de program în funcție de seturi de date inițiale simple, medii și extreme. Dacă pentru toate testele efectuate programul prezintă rezultate corecte, se poate presupune că problema a fost rezolvată corect. Dacă în procesul de testare se obțin rezultate care diferă de cele corecte, urmează ca rezolvarea problemei să fie reluată, începând cu etapa de *analiză a problemei*.

Procesul de rezolvare a unei probleme la calculator poate fi ilustrat cu ajutorul următoarei scheme:

#### Exemplu:

##### Problemă:

În condiții de laborator, o populație de viruși, formată inițial din  $N$  unități și plasată într-un mediu steril, se micșorează în fiecare oră cu 50 de procente, dacă numărul virușilor la începutul orei este par, sau crește cu o unitate, dacă numărul virușilor la începutul orei este impar. În momentul când numărul virușilor devine mai mic decât cantitatea critică de supraviețuire  $C$ , populația dispare integral.

**Cerință:** Scrieți un program care va stabili timpul necesar, în ore, pentru distrugerea în laborator a unei populații din  $N$  ( $N < 32\,000$ ) viruși, având cantitatea critică de supraviețuire  $C$  ( $1 < C < N$ ).

##### Analiza problemei

Populația formată dintr-un număr par de viruși în fiecare oră se micșorează de două ori. Populația formată dintr-un număr impar crește cu o unitate și se transformă într-o populație pară. Creșterea repetată este imposibilă, înjumătățirea se repetă însă cel puțin o dată la două ore. Prin urmare, valoarea  $C$  ( $C > 1$ ) va fi atinsă într-un număr finit de ore.

##### Modelul matematic

Numărul populației în momentul de timp  $t = 0$  este dat de numărul  $N_0 = N$ .

Numărul populației după  $t$  ore de sterilizare ( $t > 0$ ) este dat de formula recurrentă:

$$N_t = \begin{cases} \frac{N_{t-1}}{2}, & \text{dacă } N_{t-1} \text{ este par} \\ N_{t-1} + 1, & \text{dacă } N_{t-1} \text{ este impar.} \end{cases}$$

##### Algoritm

**Pasul 0. Initializare:** Se introduc valorile  $N$ ,  $C$ .  $t \leftarrow 0$ .\*

##### Pasul 1.

a) Trecerea unei ore:  $t \leftarrow t+1$ .

b) Remodelarea populației: dacă  $N \bmod 2 = 0$ ,  $N \leftarrow N \div 2$ , altfel  $N \leftarrow N+1$ .

**Pasul 2. Verificarea condiției de supraviețuire:** dacă  $N < C$ , se afișează valoarea  $t$ . SFARȘIT. În caz contrar, se revine la *pasul 1*.

\* Aici și în continuare expresia  $a \Leftarrow b$  semnifică:  $a$  primește valoarea lui  $b$ .

**Program**

```
program cn04;
var N,C,t: integer;
begin
  readln(N, C); t:=0;
  while (N>=C) do begin
    t:=t+1;
    if N mod 2 = 1 then N:=N+1 else N:= N div 2;
  end;
  writeln(t);
end.
```

```
// program cn04;
#include <iostream>
using namespace std;
int N, C, t;

int main()
{
  cin >> N >> C;
  t = 0;
  while ( N >= C )
  {
    t++;
    if (N % 2 == 1) N++; else N = N / 2;
  }
  cout << t;
  return 0;
}
```

**Testare**

Pentru verificarea corectitudinii rezolvării problemei, au fost folosite următoarele seturi de date:

Date inițiale	Rezultat						
512 2	9	31999 16	15	331 330	2	332 330	1
25768 235	10	31999 2	18	31997 2	19	3 2	3

## Întrebări și exerciții

- 1 Explică!** Enumerați etapele rezolvării unei probleme la calculator. Explicați necesitatea fiecărei etape.
- 2 Sistematizează-ți cunoștințele!** Care sunt metodele de descriere a algoritmului rezolvării unei probleme? Exemplificați.
- 3 Analizează!** Care este impactul divizării unei probleme în subprobleme elementare? Dați exemple de probleme ce pot fi divizate în subprobleme. Indicați două sau mai multe probleme ce conțin subprobleme identice.
- 4 Aplică!** Pentru următoarele probleme realizați modelul matematic:
  - a) Sunt cunoscute coordonatele a trei vârfuri ale unui dreptunghi cu laturile paralele axelor de coordinate. Se cere determinarea coordonatelor celui de-al patrulea vârf.
  - b) În condițiile punctului precedent, laturile dreptunghiului pot fi poziționate arbitrar față de axele de coordinate.
- 5 Elaborează!** Pentru următoarele probleme elaborați un algoritm de rezolvare, folosind metodele de descriere cunoscute:
  - a) Este dat un sir din cel mult 100 de numere întregi. Se cere să se aranjeze elementele sirului în ordine crescătoare.
  - b) Este dat un sir din cel mult 100 de numere întregi. Se cere determinarea elementului cu valoare maximă din sir și a numărului de repetări ale lui printr-o singură parcursere a sirului.
- 6 Programează!** Elaborați programe pentru rezolvarea problemelor din exercițiile 4 și 5.
- 7 Integrează și aplică!** Alcătuți seturi de teste pentru verificarea programelor elaborate în exercițiul 6.

### 3.5 Numere aproximative. Eroarea absolută și eroarea relativă

Numărul  $a$  se numește aproximare a numărului  $A$  dacă valorile lor se deosebesc neînsemnat și  $a$  poate înlocui  $A$  în calcule numerice. Dacă este adevărată relația  $a < A$ ,  $a$  este numit aproximare prin lipsă, dacă  $a > A$  – aproximare prin adăos. De exemplu, pentru  $\pi$  numărul 3,14 va fi aproximare prin lipsă, iar 3,142 – aproximare prin adăos. Într-adevăr,  $3,14 < \pi < 3,142$ . Valorile aproximative apar în procesul măsurărilor realizate, iar diferența valorii aproximative de cea exactă se poate datora unei varietăți de factori: condițiilor de temperatură, presiunii, umidității, calității instrumentelor de măsurare, calificării persoanei care execută măsurarea etc.

Prin eroare  $\Delta_a$  se înțelege diferența  $A-a$  (uneori și  $a-A$ ). În funcție de valorile  $a$  și  $A$ ,  $\Delta_a$  poate fi negativă sau pozitivă. Pentru a obține numărul exact  $A$ , se adaugă la  $a$  valoarea erorii  $\Delta_a$ :

$$A = a + \Delta_a.$$

De multe ori este cunoscută valoarea aproximativă  $a$  fără a se cunoaște semnul erorii. În aceste cazuri se utilizează *eroarea absolută*, care se definește în felul următor:

**Eroare absolută  $\Delta$  a valorii aproximative  $a$  se consideră modulul diferenței dintre valoarea exactă  $A$  și valoarea aproximativă  $a$ :**

$$\Delta = |A - a|.$$

Eroarea absolută nu este un indice suficient pentru estimarea exactității calculelor sau a măsurărilor. Fie că în urma măsurării lungimii a două bare, cu lungimi de 20 m și, respectiv, 6 m, au fost obținute rezultatele de măsurare 20,5 m și 6,2 m.

Cu toate că valoarea absolută a erorii în primul caz este mai mare, este evident că prima măsurare a fost realizată mult mai exact decât a doua. Pentru a determina calitatea măsurării (a calculului), se raportează mărimea erorii absolute la o unitate de lungime (sau, în caz general, la o unitate de măsură respectivă).

**Eroare relativă  $\delta$  a valorii aproximative se consideră raportul dintre eroarea absolută  $\Delta$  și modulul numărului exact  $A$  ( $A \neq 0$ ):**

$$\delta = \frac{\Delta}{|A|}.$$

**Exemplu:** O bară are lungimea de 100 cm. În urma unei măsurări a fost stabilită o valoare a lungimii egală cu 101 cm. Distanța dintre punctele  $A$  și  $B$  este de 3 000 m. Ca rezultat al măsurării s-a obținut distanța de 2 997 m. Se cere determinarea erorii relative a fiecărei măsurări și măsurarea mai exactă.

**Rezolvare:**

pentru prima măsurare  $\Delta = |100 - 101| = 1$  cm,  $\delta = \frac{1}{100} = 0,01$ ;

pentru măsurarea a doua  $\Delta = |3\ 000 - 2\ 997| = 3$  m,  $\delta = \frac{3}{3\ 000} = 0,001$ .

Deoarece eroarea relativă a celei de-a doua măsurări este mai mică, această măsurare este mai exactă.

### Info+

Reprezentarea numerelor în calculator implică utilizarea numerelor aproximative cu un număr finit de cifre. Cifrele reținute se numesc *cifre semnificative*, prima fiind obligatoriu diferită de 0.

**Exemplu:** 0,04708 are patru cifre semnificative. Primele două zero-uri doar fixează poziția virgulei zecimale.

Aproximarea  $a$  ( $a_n a_{n-1} a_{n-2} \dots a_0$ ) a numărului exact  $A$  are  $k$  cifre semnificative exacte  $a_i a_{i-1} \dots a_{i-k+1}$  dacă:

$$|A - a| \leq \frac{1}{2} 10^{i-k+1}.$$

## Întrebări și exerciții

- 1 **Explică!** Dați exemple de apariție a erorilor în situații reale.
- 2 **Explică!** Definiți noțiunea de eroare absolută. Exemplificați.
- 3 **Analizează!** Exemplificați. Cum se va modifica formula de calcul a erorii relative, dacă eroarea trebuie indicată în % de la valoarea exactă a mărimii cercetate?
- 4 **Aplică!** Lungimea traseului dintre două localități, afișată pe indicatorul de drum, este de 230 km. Parcugând acest traseu cu autovehiculul, ați fixat variația indicațiilor dispozitivului de măsurare a distanței – 230,7 km. Considerând datele indicatorului ca fiind exacte, determinați eroarea absolută și eroarea relativă a măsurării.

- 5** **Aplică!** O bară cu lungimea exactă de 100 cm a fost măsurată cu o eroare absolută de 2 cm. Care sunt valorile posibile obținute în procesul de măsurare?
- 6** **Aplică!** Volumul exact al unui vas este de 20 l. Măsurările de volum au fost efectuate cu o eroare relativă de 0,001. Care sunt valorile posibile ale volumului măsurat?

### 3.6

## Sursele erorilor de calcul

Erorile care apar în timpul rezolvării problemelor pot proveni din diferite surse. Cunoașterea surselor de apariție a erorilor permite ocolirea lor și minimizarea efectului cumulativ al erorilor. Cele mai des întâlnite tipuri de erori sunt:

- erori de problemă;
- erori de metodă;
- erori ale datelor de intrare;
- erori de aproximare;
- erori de rotunjire.

**Erori de problemă.** Această categorie de erori apare în situațiile când modelul matematic ales pentru rezolvarea problemei nu descrie complet procesul real cercetat. Astfel, în exemplul 2 (3.2. Modelul matematic și modelarea matematică), pentru a construi modelul matematic al sistemului *bilă-arc*, s-a utilizat ecuația oscilațiilor armonice pentru *deformații mici și în lipsa forței de frecare*. Prin urmare, rezultatul obținut prin utilizarea formulei va fi diferit de cel exact, diferența fiind cu atât mai semnificativă, cu cât e mai mare forța de frecare și deformarea arcului în sistemul real.

**Erori de metodă.** Este o categorie de erori generate de imposibilitatea determinării unei metode exacte de rezolvare a problemei sau de restricțiile care impun utilizarea unei metode mai puțin exacte. În acest caz problema inițială este rezolvată printr-o metodă euristică, care poate genera diferențe esențiale dintre rezultatul calculat și cel exact.

Un exemplu elocvent este utilizarea metodei *Greedy* pentru rezolvarea problemei *rucsacului*.

**Erori ale datelor de intrare.** Deseori procesul de modelare matematică se bazează pe rezultatele unor experiențe, adică pe niște seturi de mărimi numerice, obținute în urma măsurărilor. Aceste mărimi nu sunt exacte (ex.: distanță, masa, viteză).

Fie că un corp se mișcă pe o traекторie descrisă pe segmentul  $[0,1]$  de funcția  $f(x) = x^2 + x + 1$ . Se știe că valoarea argumentului  $x$  se calculează cu o eroare absolută care nu depășește 0,01. Prin urmare, dacă  $z$  este valoarea exactă a argumentului, atunci  $|z - x| < 0,01$ .

### Ne amintim!

#### Problema rucsacului:

Fie un rucsac de volum  $S$  și  $n$  obiecte de volume  $v_i$ ,  $i = 1 \dots n$  și costuri  $c_i$ ,  $i = 1 \dots n$ . Se cere să se pună în rucsac obiecte din setul propus astfel încât costul total al obiectelor puse în rucsac să fie maxim posibil.

Dacă pentru rezolvare este folosită metoda *Greedy*, în majoritatea cazurilor soluția obținută va fi diferită de cea optimă. Astfel, pentru un set din cinci obiecte cu volumele 5, 7, 13, 20, 10, costurile respective de 4, 8, 15, 23, 5 și un rucsac cu volumul de 30 de unități va genera soluția cu valoarea 27 (obiectele 3, 2, 1). (Sortarea obiectelor este realizată în descreșterea raportului cost/volum.) În realitate, soluția optimă are valoarea 31 (obiectele 2 și 4).

În condițiile date se poate stabili în ce măsură eroarea la măsurarea valorilor lui  $x$  influențează rezultatul calculului:

$$\begin{aligned}|f(x) - f(z)| &= |x^2 + x + 1 - z^2 - z - 1| = \\&= |x^2 + x - z^2 - z| = |(x-z)(x+z+1)|.\end{aligned}$$

Deoarece  $x+z+1 \leq 3$  și  $|z-x| < 0,01$ , rezultă  $|f(x) - f(z)| \leq 0,03$ .

Diferența dintre valoarea funcției de argument exact ( $z$ ) și valoarea funcției de argument măsurat ( $x$ ) este o mărime constantă. De aici reiese că erorile de calcul nu depind de  $x$ , ci numai de exactitatea cu care acesta este măsurat, iar funcția este stabilă la erori.

**Erori de aproximare.** Este o categorie de erori generate de anumite definiții și noțiuni matematice. Prezența lor este acceptată în special în problemele care folosesc noțiunea de limită, convergență etc. Apariția acestui tip de erori este motivată de însăși structura definiției care conține elemente de aproximare.

**Exemplu:** Sirul  $\{x_n\}$  este convergent și are limita  $x$  dacă pentru orice  $\varepsilon > 0$ ,  $\varepsilon \rightarrow 0$ , există un rang  $n(n_\varepsilon)$ , astfel încât  $\forall n > n_\varepsilon$ ,  $|x - x_n| < \varepsilon$ .

Din punctul de vedere al analizei numerice,  $\varepsilon$  indică precizia cu care  $x_n$  aproximează  $x$ .

În procesul de calcul al limitei se utilizează o aproximare consecutivă, care se apropiă tot mai mult de limita exactă, fără să atinge. Stoparea procesului de calcul are loc atunci când deviația (eroarea) devine mai mică decât eroarea maximă admisibilă ( $\varepsilon$ ).

De reținut că rezultatele obținute prin calcul numeric sunt acceptate doar ca rezultate cunoscătoare și nu pot servi drept demonstrație pentru anumite afirmații matematice.

**Erori de rotunjire.** Este un tip aparte de erori, generate de faptul că în procesul prelucrării mărimilor numerice în calculator ele pot fi păstrate doar cu un anumit număr de semne zecimale după virgulă. Drept exemplu poate servi constanta  $\pi$ , valorile funcțiilor trigonometrice etc.

Fie  $A = \{a_1, a_2, a_3, \dots, a_n\}$  mulțimea tuturor numerelor care pot fi reprezentate în calculator. (Se consideră că  $a_1 < a_2 < a_3 < \dots < a_n$ .)

### Ne amintim!

Se consideră că funcția  $f: I \rightarrow R$  posedă proprietatea Lipschits dacă există o constantă  $m > 0$ , astfel încât:  $|f(x) - f(z)| \leq m^* |z-x|$ ,  $\forall x, z \in I$ .

Se consideră că determinarea valorilor funcției  $f(x)$  este stabilă la erori, dacă  $f(x)$  posedă proprietatea Lipschits.

În alți termeni, stabilitatea funcției la erori presupune variații mici ale valorilor funcției la variații mici ale argumentului.

### Ne amintim!

Apropierea valorilor termenilor sirului  $1/n$  de limita sirului - 0.

$n = 1$	$1/n = 1$
$n = 2$	$1/n = 0,5$
...	
$n = 501$	$1/n = 0,001996$
$n = 502$	$1/n = 0,001992$
$n = 503$	$1/n = 0,001988$
...	
$n = 999$	$1/n = 0,001001$
$n = 1000$	$1/n = 0,001000$
$n = 1001$	$1/n = 0,000999$
...	

**Exemplu:** Fie că în procesul de calcul se poate opera cu cel mult 3 cifre după virgulă. În acest caz pentru numerele

$$a = 0,2334, b = 0,2331, c = 0,233$$

$$\begin{aligned}\text{se obține} \quad rd(rd(a+b)+c) &= \\&= rd(rd(0,4665)+0,233) = \\&= rd(0,467+0,233) = \mathbf{0,7}; \\rd(a+rd(b+c)) &= \\&= rd(0,2334+rd(0,4661)) = \\&= rd(0,2334+0,466) = \\&= rd(0,6994) = \mathbf{0,699}.\end{aligned}$$

Oricare dintre numerele  $\frac{a_i + a_{i+1}}{2}$  lipsește în mulțimea inițială  $A$ . În cazul apariției unei asemenea situații de calcul, apare eroarea legată de înlocuirea rezultatului real prin numărul cel mai apropiat de el din mulțimea  $A$ . Această procedură este numită *rotunjire*.

*Funcția de rotunjire* aplicată la calculatoare se definește în felul următor:

$$\forall x \in R, a_i \in A, a_{i+1} \in A \quad x \in (a_i, a_{i+1}),$$

$$rd(x) = a_i \text{ dacă } x \in \left( a_i, \frac{a_i + a_{i+1}}{2} \right),$$

$$rd(x) = a_{i+1} \text{ dacă } x \in \left[ \frac{a_i + a_{i+1}}{2}, a_{i+1} \right).$$

Utilizarea funcției de rotunjire generează abateri de la legile de bază ale operațiilor aritmétice, care nu mai sunt asociative, distributive.

În cazul rotunjirii rezultatelor de calcul, erorile nu depășesc după modul valoarea de  $0,5 \cdot 10^{-n}$  (aceste erori se numesc *erori de rotunjire absolute*), unde  $n$  este numărul de semne semnificative (care pot fi percepute) în procedura de calcul. Erorile de rotunjire pot fi atât pozitive, cât și negative. În cazul alternării lor, are loc procesul de compensare, ca rezultat, eroarea finală nu crește odată cu numărul de calcule.

## Întrebări și exerciții

- 1 **Sistematizează-ți cunoștințele!** Care sunt principalele surse de erori? Exemplificați.
- 2 **Aplică!** Dați exemple de erori generate de imposibilitatea formulării exacte a problemei. Motivați imposibilitatea formulării exacte.
- 3 **Analizează!** Care funcții posedă proprietatea de stabilitate a calculului valorilor față de erori?
- 4 **Aplică!** Este oare stabil față de erori calculul valorilor funcției liniare? Dar al funcției  $y = \sqrt{x}$ ,  $x \in [1,2]$ ?
- 5 Explicați esența erorilor de aproximare.
- 6 **Studiu de caz!** Soluția unei probleme se calculează iterativ după formula:  $x_0 = 0$ ,  $x_{i+1} = \sqrt{2 + x_i}$ . Va atinge oare sirul soluțiilor calculate valoarea soluției exacte? Stabiliti experimental sau analitic soluția exactă. După câte iterații eroarea absolută a soluției calculate va deveni mai mică de 0,0001?
- 7 **Explică!** Din care motiv rezultatele obținute prin metode numerice nu pot servi drept demonstrații ale teoremelor matematice?
- 8 **Explică!** Care este cauza apariției erorilor de rotunjire? Exemplificați.
- 9 **Experimentează!** Elaborați un program care determină produsul și câtul numerelor 1,00000001 și 0,99999999. Valorile vor fi stocate în variabile, având tipul **real**. Analizați rezultatele obținute. Explicați cauzele apariției erorii.

## 3.7 Separarea soluțiilor ecuațiilor algebrice și transcendentale

A rezolva ecuația algebrică sau transcendentală (în continuare *ecuația*)  $f(x) = 0$  înseamnă a determina acele valori ale variabilei  $x$  pentru care egalitatea  $f(x) = 0$  este una adevărată. În cazul în care ecuația are o structură simplă, soluțiile ei pot fi determinate exact și relativ ușor prin metodele analitice, care se studiază în cadrul cursului liceal de matematică. Dacă însă structura ecuației este complicată, procedura de determinare a soluțiilor devine destul de anevoieoașă. Mai mult decât atât, atunci când ecuația modeleză anumite situații, fenomene care depind de mai mulți parametri, iar valoarea acestora este cunoscută doar aproximativ, noțiunea de soluție exactă în general își pierde sensul. Din acest motiv, este util de a cunoaște și metodele de calcul aproximativ al soluțiilor ecuațiilor și algoritmii care realizează aceste metode.

Fie dată ecuația

$$f(x) = 0, \quad (1)$$

$f(x)$  fiind definită și continuă pe un oarecare interval  $a \leq x \leq b$ .

**Orice valoare  $\xi$ , pentru care expresia  $f(\xi) = 0$  este adevărată, se numește zero al funcției  $f(x)$  sau soluție a ecuației  $f(x) = 0$ .**

În cele ce urmează se va presupune că ecuația (1) are soluții distințe (izolate), adică pentru fiecare soluție a ecuației există o vecinătate a sa, care nu conține alte soluții.

Astfel, rezolvarea prin metode numerice a unei ecuații se divide în două etape:

1. Separarea intervalelor pe care ecuația are o singură soluție.

2. Micșorarea pe cât mai mult posibil a fiecărui dintre aceste intervale (dacă se pune problema determinării tuturor soluțiilor) sau a unuia dintre ele (dacă trebuie de determinat doar una dintre soluții).

**Metoda analitică.** Pentru separarea analitică a soluțiilor, vor fi folosite proprietățile derivatei.

Dacă soluțiile ecuației  $f'(x)=0$  pot fi ușor calculate, atunci, pentru a separa soluțiile  $f(x)=0$ , este necesar:

1. să se determine soluțiile distințe  $a \leq x_1 \leq x_2 \leq \dots \leq x_n \leq b$  ale ecuației  $f'(x)=0$ ;
2. considerând  $a = x_0$  și  $b = x_{n+1}$ , să se calculeze valorile  $f(x_0), f(x_1), \dots, f(x_{n+1})$ . Segmentele  $[x_i, x_{i+1}], i = 0, \dots, n$ , pentru care  $f(x_i) \times f(x_{i+1}) < 0$  vor conține câte cel puțin o soluție a ecuației  $f(x)=0$ .

**Exemplul 1:** Să se determine numărul de soluții ale ecuației  $e^x + x = 0$ :

$$f'(x) = e^x + 1; \quad f'(x) > 0 \quad \forall x \in \mathbb{R}.$$

### Info+

Dacă funcția  $f(x)$  are forma unui polinom sau poate fi adusă la această formă, ecuația  $f(x) = 0$  se numește **algebrică**.

**Exemplu:**  $3x - 7 = 0$ .

În caz contrar – când  $f(x)$  nu este una polinomială, ecuația se numește **transcendentă**.

**Exemplu:**  $\sin(2x) + \sqrt{\cos^2 x + e^x} = 0$ .

### Ne amintim!

**Teoremă.** Dacă funcția  $f(x)$ , continuă pe segmentul  $[a, b]$ , primește la extremitățile lui valori de semn diferit ( $f(a) \times f(b) < 0$ ), atunci pe acest segment există cel puțin un punct  $\xi$ , astfel încât  $f(\xi) = 0$ . Dacă pe  $[a, b]$  există derivata  $f'(x)$ , continuă, care are un semn constant, atunci  $\xi$  este soluția unică a ecuației  $f'(x) = 0$  pe acest segment.

Întrucât  $\lim_{x \rightarrow -\infty} f(x) = -\infty$ ,  $\lim_{x \rightarrow \infty} f(x) = \infty$ , ecuația inițială are o singură soluție.

**Exemplul 2:** Să se separe rădăcinile ecuației  $x^3 - 9x^2 + 24x - 19 = 0$  pe segmentul  $[0, 8]$ .

Rezolvare:

$$f(x) = x^3 - 9x^2 + 24x - 19;$$

$$f'(x) = 3x^2 - 18x + 24.$$

Rezolvând ecuația  $f'(x) = 0$ , se obțin soluțiile  $x_1 = 2$ ,  $x_2 = 4$ .

x	f(x)	Semn f(x)
0	-19	-
2	1	+
4	-3	-
8	109	+

Deci ecuația va avea trei soluții, câte una pe fiecare dintre segmentele  $[0, 2]$ ,  $[2, 4]$ ,  $[4, 8]$ .

**Metoda grafică.** O altă posibilitate de separare a rădăcinilor ecuației  $f(x) = 0$  este cercetarea directă a graficului funcției  $f(x)$ . Pentru construcția acestuia pot fi folosite atât aplicații software specializate\*, cât și programe simple, elaborate cu ajutorul instrumentelor unui limbaj de programare.

Separarea grafică a soluțiilor unei ecuații pe un segment dat poate fi realizată și local, cu ajutorul unei aplicații de calcul tabelar. Este suficient să se construiască un tabel cu două coloane. Prima coloană va reprezenta o divizare a segmentului în segmente elementare de lungimi egale. Cea de-a doua coloană va conține o formulă care calculează valoarea funcției  $f(x)$  pentru valorile respective din prima coloană. În baza datelor din coloana cu valorile  $f(x)$  se construiește o diagramă liniară, care reprezintă graficul funcției analizate.

**Exemplu:**  $f(x) = x^{\cos(2x)} + 3 \sin(x)$ . Soluțiile se caută pe segmentul  $[0, 2, 10]$  (fig. 3.6a, 3.6b).

A	B	C	D	E	F
1 x	y				
2 0,2	0,823102167				
3 0,4	1,696399241				
4 0,6	2,524947747				
46	5,503155521				
47 9,2	8,048154414				
48 9,4	9,448504359				
49 9,6	7,844007308				
50 9,8	4,209027748				
51 10	0,927006056				

Fig. 3.6a. Funcția  $f(x)$  reprezentată tabelar în foaia de calcul. Coloana A conține valorile  $x$  de la 0,2 la 10 cu pasul 0,2. Coloana B conține valorile  $f(x)$

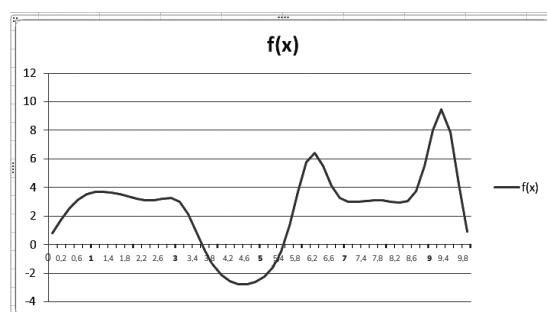


Fig. 3.6b. Funcția  $f(x)$  reprezentată grafic în baza datelor din tabel. Pot fi ușor identificate două segmente oarecare, ce conțin exact câte o soluție a ecuației  $f(x)=0$ , de exemplu:  $[3, 5]$  și  $[5, 6]$

\* MatCAD, Mathematica, Derive etc.

## Întrebări și exerciții

- 1** Ce numim soluție a unei ecuații?
- 2** **Explică!** Ce condiții trebuie să satisfacă funcția  $f(x)$  pentru ca pe un segment dat să fie cel puțin o rădăcină a ecuației  $f(x)=0$ ? Dar pentru existența exact a unei soluții?
- 3** **Exersează!** Determinați analitic numărul de soluții reale ale ecuației:  
 a)  $x^5 - 5x + 7 = 0$ ;      b)  $x^3 - 9x^2 + 24x - 13 = 0$ .
- 4** **Programează!** Elaborați un program care pentru funcția  $f(x)$ , continuă pe  $[a, b]$ , realizează o divizare a segmentului în  $n$  segmente de lungime egală și afișează toate segmentele la extremitățile cărora funcția are valori de semn opus:  
 a)  $f(x) = x^3 - 7x^2 + 12x - 37$       pe  $[-10, 10]$ ,  $n = 100$ ;  
 b)  $f(x) = \sin(3x) + 3 \cos(x) - 1$       pe  $[-2, 2]$ ,  $n = 100$ .
- 5** **Aplică!** Separați, folosind cea mai potrivită metodă, soluțiile ecuațiilor:  
 a)  $\frac{x^4}{4} + x^3 - \frac{x^2}{2} - 3x - 8 = 0$ ;      e)  $e^x(x^2 - 2x + 2) = 0$ ;  
 b)  $2x^3 - 6x^2 - 48x + 17 = 0$ ;      f)  $x[\sin(\ln(x)) - \cos(\ln(x))] = 0$ ;  
 c)  $x^4 - 14x^2 - 24x - 4 = 0$ ;      g)  $\ln(x) + \sqrt{\sin(3x) + 7} - x^2 = 0$ .  
 d)  $\frac{1}{2} - e^{-x^2} = 0$ ;

### 3.8 Metoda bisecției

Fie dată funcția  $f(x)$ , continuă pe segmentul  $[a, b]$ , și  $f(a) \times f(b) < 0$ .

Se cere să se determine pe segmentul  $[a, b]$  o soluție a ecuației

$$f(x) = 0. \quad (1)$$

Proprietățile funcției asigură existența a cel puțin unei soluții pe acest segment.

Una dintre cele mai simple metode de determinare a unei soluții a ecuației  $f(x) = 0$  este *metoda bisecției*. Metoda presupune determinarea punctului de mijloc  $c$  al segmentului  $[a, b]$ , apoi calculul valorii  $f(c)$ . Dacă  $f(c) = 0$ , atunci  $c$  este soluția exactă a ecuației. În caz contrar, soluția este căutată pe unul dintre segmentele  $[a, c]$  și  $[c, b]$ . Ea va apartine segmentului pentru care semnul funcției în extremități este diferit (fig. 3.7).

Dacă  $f(a) \times f(c) > 0$ , atunci soluția e căutată în continuare pe segmentul  $[a_1, b_1]$ , unde  $a_1$  primește valoarea  $c$ , iar  $b_1$  – valoarea  $b$ . În caz contrar,  $a_1$  primește valoarea  $a$ , iar  $b_1$  – valoarea  $c$ . Procesul de divizare se reia pe segmentul  $[a_1, b_1]$ , repetându-se până când se obține soluția exactă sau (în majoritatea absolută a cazurilor!) devierea soluției calculate  $c_i$  de la cea exactă devine suficient de mică.

În urma divizărilor succesive, se obține consecutivitatea segmentelor

$$[a_0, b_0], [a_1, b_1], [a_2, b_2], \dots, [a_i, b_i], \dots .$$

Pentru fiecare dintre ele are loc relația

$$f(a_i) \times f(b_i) < 0, i = 0, 1, 2, \dots . \quad (2)$$

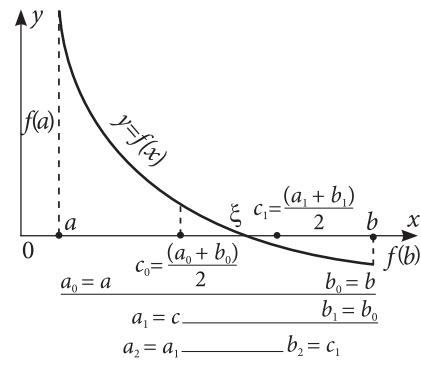


Fig. 3.7. Calculul consecutiv al segmentelor care conțin soluția ecuației  $f(x)=0$

**Estimarea erorii.** Deoarece soluția exactă  $\xi$  a ecuației este un punct al segmentului  $[a_i, b_i]$ , rezultă că diferența dintre soluția exactă și cea calculată nu depășește lungimea acestui segment. Prin urmare, localizarea soluției pe un segment cu lungimea  $\varepsilon$  asigură o eroare de calcul a soluției ce nu depășește valoarea  $\varepsilon$ :

$$|\xi - c_i| < \varepsilon = |b_i - a_i|.$$

### Algoritmizarea metodei

Pornind de la descrierea matematică a metodei, putem separa două cazuri distincte de oprire a procesului de calcul al soluției ecuației  $f(x) = 0$  pentru metoda bisecției:

#### A1. Algoritmul de calcul pentru un număr prestabilit $n$ de divizări consecutive:

**Pasul 0.** Inițializare:  $i \Leftarrow 0$ .

**Pasul 1.** Determinarea mijlocului segmentului  $c \Leftarrow \frac{a+b}{2}$ .

**Pasul 2.** Reducerea segmentului ce conține soluția: dacă  $f(c) = 0$ , atunci soluția calculată este  $x = c$ . SFARȘIT.

În caz contrar, dacă  $f(a) \times f(c) > 0$ , atunci  $a \Leftarrow c$ ;  $b \Leftarrow b$ , altfel  $a \Leftarrow a$ ;  $b \Leftarrow c$ .

**Pasul 3.**  $i \Leftarrow i + 1$ . Dacă  $i = n$ , atunci soluția calculată este  $x = \frac{a+b}{2}$ . SFARȘIT.

În caz contrar, se revine la *pasul 1*.

#### A2. Algoritmul de calcul pentru o precizie\* $\varepsilon$ dată:

**Pasul 1.** Determinarea mijlocului segmentului  $c \Leftarrow \frac{a+b}{2}$ .

**Pasul 2.** Dacă  $f(c) = 0$ , atunci soluția calculată este  $x = c$ . SFARȘIT.

În caz contrar, dacă  $f(a) \times f(c) > 0$ , atunci  $a \Leftarrow c$ ;  $b \Leftarrow b$ , altfel  $a \Leftarrow a$ ;  $b \Leftarrow c$ .

**Pasul 3.** Dacă  $|b - a| < \varepsilon$ , atunci soluția calculată este  $x = \frac{a+b}{2}$ . SFARȘIT.

În caz contrar, se revine la *pasul 1*.

**Exemplul 1:** Să se determine o rădăcină a ecuației  $x^4 + 2x^3 - x - 1 = 0$  pe segmentul  $[0, 1]$  pentru 16 divizări consecutive.

Deoarece numărul de aproximări succesive este fixat, iar extremitățile segmentului cunoscute, atribuirile se realizează nemijlocit în program.

```
program cn05;
var a,b,c: real;
    i,n:integer;

function f(x:real):real;
begin  f:=sqr(sqr(x))+2*x*sqr(x)-x-1;end;
begin  a:=0; b:=1; n:=16;
      for i:=1 to n do
        begin  c:=(b+a)/2;
               writeln('i=',i:3,' x=',c:10:8,' f(x)=',f(c):12:8);
               if f(c)=0 then break1
               else if f(c)*f(a)>0 then a:=c else b:=c;
        end;
end.
```

\* În contextul dat precizia  $\varepsilon$  semnifică o eroare de calcul, care nu depășește valoarea  $\varepsilon$ .

<sup>1</sup> Instrucțiune de salt necondiționat care întrerupe execuția instrucțiunii ciclice în care se conține.

*Rezultate:*

```
i= 1 x=0.50000000 f(x)= -1.18750000
i= 2 x=0.75000000 f(x)= -0.58984375
...
i= 15 x=0.86679077 f(x)= 0.00018565
i= 16 x=0.86677551 f(x)= 0.00009238
```

```
// program cn05;
#include <iostream>
#include <iomanip>
#include <math.h>
using namespace std;

double a, b, c;
int i, n;
double f(double x)
{
    return pow(x, 4)+2*x*pow(x, 2)-x-1;
}

int main()
{
    a = 0; b = 1; n = 16;
    for (i = 1; i <= n; i++)
    {
        c = (b + a) / 2;
        cout << setprecision(8) << "i=" << i << " x=" << c << "
f(x)="<<f(c) << endl;
        if (f(c) == 0) break;
        else if (f(c) * f(a) > 0 ) a = c; else b = c;
    }
    return 0;
}
```

*Rezultate:*

```
i=1 x=0.5 f(x)=-1.1875
i=2 x=0.75 f(x)=-0.58984375
...
i=15 x=0.86679077 f(x)=0.00018565479
i=16 x=0.86677551 f(x)=9.2381174e-05
```

**Exemplul 2:** Să se determine o rădăcină a ecuației  $6 \cos(x) + 8 \sin(x) = 0$  pe segmentul  $[2, 4]$  cu precizia  $\varepsilon=0,00017$ .

```
program cn06;
var a,b,c,eps: real;

function f(x:real):real;
```

```

begin   f:=6*cos(x)+8*sin(x);end;

begin   a:=2; b:=4; eps:=0.00017;
  repeat
    c:=(b+a)/2;
    writeln('x=',c:10:8,' f(x)=',f(c):12:8);
    if f(c)=0 then break
    else if f(c)*f(a)>0 then a:=c else b:=c;
  until abs(b-a)<eps;
end.

```

*Resultate:*

x=3.00000000	f(x)= -4.81099492
x=2.50000000	f(x)= -0.01908454
...	
x=2.49829102	f(x)= -0.00199471
x=2.49816895	f(x)= -0.00077401

```

// program cn06;
#include <iostream>
#include <iomanip>
#include <math.h>
using namespace std;

double a, b, c, eps;
int i, n;
double f(double x)
{
    return 6 * cos(x) + 8 * sin(x);
}

int main()
{
    a = 2; b = 4; eps = 0.00017;
    do
    {
        c = (b + a) / 2;
        cout << setprecision(8) << " x=" << c << " f(x)=" << f(c) << endl;
        if (f(c) == 0) break;
        else if (f(c) * f(a) > 0 ) a = c; else b = c;
    } while (abs(a - b) > eps);
    return 0;
}

```

*Resultate:*

x=3 f(x)=-4.8109949
x=2.5 f(x)=-0.01908454
...
x=2.498291 f(x)=-0.0019947083
x=2.4981689 f(x)=-0.00077400516

## Întrebări și exerciții

- 1 Explică!** În ce cazuri se folosesc metode aproximative de determinare a soluțiilor ecuațiilor algebrice?

Descrieți metoda bisecției. Care sunt prioritățile ei? Dar neajunsurile? Formula pentru estimarea erorii, dedusă în paragraful curent, este  $|\xi - c_i| < \varepsilon = |b_i - a_i|$ . Exprimă diferența dintre valoarea exactă și cea calculată printr-o formulă care depinde doar de extremitățile segmentului inițial și numărul de divizări realizate.

- 2** Descrieți algoritmul metodei bisecției.

- 3 Aplică!** Determinați prin metoda bisecției soluțiile ecuațiilor:

1.  $e^x - x^2 = 0$  pe  $[-1, -0,5]$ ;
  2.  $x^3 - x - 1 = 0$  pe  $[1, 2]$ ;
  3.  $x^3 + 3x^2 - 3 = 0$  pe  $[-3, -2]$ ;
  4.  $x^5 - x - 2 = 0$  pe  $[1, 2]$ .
- a) pentru 10, 20, 40 de divizări ale segmentului inițial;  
 b) cu precizia  $\varepsilon = 0,001; 0,0001; 0,00001$ ;  
 c) în condițiile punctului precedent, determinați numărul de divizări necesare pentru a obține precizia cerută.

- 4 Experimentează!** Determinați prin metoda bisecției soluțiile ecuațiilor de pe intervalele separate în exercițiile 3 și 5 de la pagina 105, pentru 10, 20, 30 de divizări consecutive.

### 3.9 Metoda coardelor

Metoda bisecției, cu toată simplitatea ei, nu este eficientă în cazurile când rezultatul trebuie obținut printr-un număr redus de operații, cu o exactitate înaltă. Astfel stând lucrurile, este mai potrivită *metoda coardelor*, care constă în divizarea segmentului în părți proporționale, proporția fiind dată de punctul de intersecție al coardei care unește extremitățile segmentului cu axa  $Ox$ .

Fie dată funcția  $f(x)$ , care posedă următoarele proprietăți:

1.  $f(x)$  continuă pe segmentul  $[a, b]$  și  $f(a) \times f(c) < 0$ .
2. Pe segmentul  $[a, b]$  există  $f'(x) \neq 0; f''(x) \neq 0$ , continui, iar semnul lor pe  $[a, b]$  este constant.

Proprietățile enumerate garantează existența soluției unice a ecuației  $f(x) = 0$  pe  $[a, b]$ .

Metoda coardelor presupune alegerea în calitate de aproximare a soluției punctul determinat de intersecția dreptei ce trece prin punctele  $(a, f(a))$  și  $(b, f(b))$  cu axa  $Ox$ .

Pentru realizarea metodei, se stabilește extremitatea  $e$  a segmentului  $[a, b]$  prin care se va duce o serie de coarde (fig. 3.8). Această extremitate este determinată de condiția:

$$f(e) \times f''(e) > 0.$$

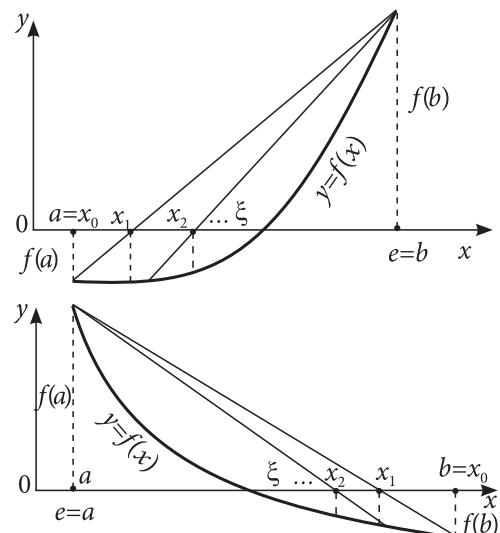


Fig. 3.8. Apropierea succesivă de soluția ecuației prin metoda coardelor

Cealaltă extremitate a segmentului  $[a, b]$  se consideră aproximare inițială a soluției:  $x_0$ .

Prin punctele  $(e, f(e))$  și  $(x_0, f(x_0))$  se construiește o coardă. Se determină punctul  $x_1$  în care coarda intersectează axa  $0x$ . Punctul  $x_1$  este considerat următoarea aproximare a soluției.

Procesul se repetă, coarda următoare fiind dusă prin punctele  $(e, f(e))$  și  $(x_1, f(x_1))$ . Astfel se obține sirul de aproximări  $x_0, x_1, x_2, \dots, x_i, x_{i+1}, \dots, x_n, \dots$ , limita căruia este soluția exactă a ecuației  $f(x) = 0$ .

Punctele  $e$  și  $x_0$  sunt cunoscute. Folosind ecuația dreptei ce trece prin două puncte, putem determina aproximarea  $x_1 (f(x_1) = 0)$ :

$$\frac{x - x_0}{e - x_0} = \frac{y - f(x_0)}{f(e) - f(x_0)},$$

de unde

$$x_1 = x_0 - \frac{f(x_0)}{f(e) - f(x_0)}(e - x_0).$$

În general, având calculată aproximarea  $x_{i-1}$ , putem determina următoarea aproximare  $x_i$  prin formula recurrentă:

$$x_i = x_{i-1} - \frac{f(x_{i-1})}{f(e) - f(x_{i-1})}(e - x_{i-1}), \quad (3)$$

$$i = 1, 2, \dots.$$

Se demonstrează că sirul de valori  $x_1, x_2, \dots, x_i, x_{i+1}, \dots, x_n, \dots$  calculate după formula (3) converge către soluția  $\xi$  a ecuației  $f(x) = 0$ .

### Eroarea metodei

Faptul că sirul aproximărilor succesive prin metoda coardelor converge către soluția exactă implică următoarea concluzie: eroarea soluției calculate va fi invers proporțională cu numărul de operații efectuate.

### Algoritmizarea metodei

Aplicarea metodei coardelor necesită o cercetare prealabilă a funcției  $f(x)$ , pentru stabilitatea extremității fixe, din care vor fi traseate coardele. Numărul  $n$  de aproximări succesive ale soluției poate fi indicat în enunțul problemei sau determinat de o condiție. În ambele cazuri calculul se realizează conform formulei (3). Condiția de oprire va fi aplicarea repetată de  $n$  ori a formulei (3).

**Determinarea extremității fixe.** Pentru a evita calculul  $f''(x)$ , se va folosi următorul procedeu: se determină semnul  $f(x)$  în punctul  $c$  de intersecție cu axa  $0x$  al dreptei care trece prin punctele  $(a, f(a))$  și  $(b, f(b))$ . **Fixă va fi acea extremitate  $e$  a segmentului  $[a, b]$ , pentru care se îndeplinește condiția:  $f(e) \times f(c) < 0$ .**

#### A1. Algoritmul de calcul pentru un număr prestabilit $n$ de aproximări successive:

**Pasul 1.** Determinarea extremității fixe  $e$  și a aproximării  $x_0$ :

$$c \Leftarrow a - \frac{f(a)}{f(b) - f(a)}(b - a);$$

dacă  $f(c) \times f(a) < 0$ , atunci  $e \Leftarrow a$ ,  $x_0 \Leftarrow b$ , altfel  $e \Leftarrow b$ ,  $x_0 \Leftarrow a$ ;  $i \Leftarrow 0$ .

**Pasul 2.** Calculul  $x_{i+1}$  conform formulei  $x_{i+1} = x_i - \frac{f(x_i)}{f(e) - f(x_i)}(e - x_i)$ .

**Pasul 3.** Dacă  $i + 1 = n$ , atunci soluția calculată  $x \Leftarrow x_i$ . SFÂRȘIT.

În caz contrar,  $i \Leftarrow i+1$  și se revine la *pasul 2*.

**Exemplu:** Fie dată funcția  $f(x) = \ln(x \sin x)$ . Să se calculeze soluția aproximativă a ecuației  $f(x) = 0$  pe segmentul  $[0,5; 1,5]$  pentru 10 aproximări succesive, utilizând metoda coardelor.

Pentru acest exemplu, preprocesarea matematică nu este necesară.

Deoarece numărul de aproximări succesive este fixat, iar extremitățile segmentului cunoscute, atribuirile respective vor fi realizate direct în corpul programului.

```
program cn07;
var a,b,e,c,x: real;
n,i: integer;

function f(x:real):real;
begin f:=ln(x*sin(x));end;
begin a:=0.5; b:=1.5; n:=10;
{determinarea extremitatii fixe e si a aproximarii initiale x0}
c:=a-(f(a))/(f(b)-f(a))*(b-a);
if f(c)*f(a)>0 then begin e:=b; x:=a; end
else begin e:=a; x:=b; end;
{calculul iterativ al solutiei}
for i:=1 to n do
begin x:= x-(f(x))/(f(e)-f(x))*(e-x);
writeln(x:10:8,' ',f(x):12:8);
end;
end.
```

*Rezultate:*

i= 1	x=1.27995775	f(x)= 0.20392348
i= 2	x=1.18251377	f(x)= 0.09028687
...		
i= 9	x=1.11427651	f(x)= 0.00016577
i= 10	x=1.11420523	f(x)= 0.00006678

```
// program cn07;
#include <iostream>
#include <iomanip>
#include <math.h>
using namespace std;

double a, b, c, e, x;
int i, n;

double f(double x)
{
    return log(x * sin(x));
}

int main()
{
```

```

a = 0.5; b = 1.5; n = 10;
// determinarea extremitatii fixe e si a aproximarii initiale x0}
c = a - f(a) / (f(b) - f(a)) * (b - a);
if (f(c) * f(a) > 0) { e = b; x = a;}
else { e = a; x = b; }
// calculul iterativ al solutiei}
for (i = 1; i <= n; i++)
{
    x = x - (f(x)) / (f(e) - f(x)) * (e-x);
    cout << setprecision(8) << " x=" << x << " f(x)=" << f(x) <<
endl;
}
return 0;
}

```

*Rezultate:*

x=1.2799577	f(x)=0.20392348
x=1.1825138	f(x)=0.090286871
...	
x=1.1142765	f(x)=0.00016577026
x=1.1142052	f(x)=6.6781584e-05

## Întrebări și exerciții

- 1 Sistematizează-ți cunoștințele!** Explicați esența metodei coardelor. Descrieți metoda grafic.
- 2 Analizează!** Cum depinde extremitatea fixă de semnul  $f''(x)$ ?
- 3 Sistematizează-ți cunoștințele!** Descrieți procesul de determinare a extremității fixe. Cum poate fi omis calculul  $f''(x)$ ?
- 4 Sistematizează-ți cunoștințele!** Descrieți pe pași algoritmul metodei coardelor pentru un număr fix de operații.
- 5 Aplică!** Determinați soluțiile ecuațiilor, utilizând metoda coardelor, pentru 10, 20, 30 de operații:
  - a)  $x^3 - 0,2x^2 + 0,2x - 2,1 = 0$  pe  $[1, 2]$ ;
  - b)  $5x^3 - 20x + 3 = 0$  pe  $[0, 1]$ ;
  - c)  $e^x - x^2 = 0$  pe  $[-1, 0]$ .
- 6 Integrează!** Separați soluțiile ecuațiilor care urmează. Rezolvați ecuațiile folosind metoda coardelor:
  - a)  $\operatorname{tg}(0,55x + 0,1) - x^2 = 0$  pentru 5, 25 de operații;
  - b)  $x^3 - 0,2x^2 + 0,5x + 1,5 = 0$  pentru 3, 9, 27 de operații.
- 7 Analizează!** Determinați prin metoda coardelor soluțiile ecuațiilor de pe intervalele separate în exercițiile 3 și 5 de la pagina 105, pentru 10, 20, 30 de operații. Comparați rezultatele cu cele obținute în exercițiul 4 de la pagina 109. Explicați diferențele.
- 8 Aplică!** Fie dată funcția  $f(x) = x[\sin(\ln(x)) - \cos(\ln(x))]$ . Determinați soluția ecuației  $f(x) = 0$  pe segmentul  $[2, 3]$  pentru 10, 20, 30 de operații, utilizând metoda coardelor.

### 3.10 Metoda dreptunghiurilor pentru calculul aproximativ al integralei definite

Una dintre cele mai des aplicate implementări ale calculului numeric este calcularea integralei definite prin metode aproximative. Metodele directe nu întotdeauna permit calculul analitic al integralei, de multe ori nici nu este cunoscută formula care definește funcția ce urmează să fie integrată. De obicei, este dată doar o serie de puncte în care este cunoscută valoarea funcției. În aceste cazuri, integrala poate fi calculată doar prin metode aproximative (în presupunerea că funcția integrată este continuă pe segmentul pe care se face integrarea).

Din cursul de matematică se știe că sensul geometric al integralei definite  $\int_a^b f(x)dx$  este aria trapezului curbiliniu, determinat de axa  $0x$ , dreptele  $x = a$ ,  $x = b$  și graficul funcției  $f(x)$  pe segmentul  $[a, b]$  (fig. 3.9).

Măsurarea exactă a ariei unei asemenea figuri nu este întotdeauna posibilă. În aceste situații o soluție ar fi aproximarea figurii inițiale cu un set de figuri, a căror arie este ușor de calculat. Atunci valoarea integralei definite (aria figurii inițiale) va fi aproximată de suma ariilor calculate.

Fie  $f$  o funcție derivabilă pe  $[a, b]$  și se cere să se calculeze  $\int_a^b f(x)dx$ .

Pentru rezolvarea numerică a problemei, se consideră o diviziune a  $[a, b]$  în forma  $a = x_0 < x_1 < \dots < x_{n-1} < x_n = b$ . Punctele consecutive de puncte  $x_0, x_1, x_2, \dots, x_i, x_{i+1}, \dots, x_{n-1}, x_n$  determină  $n$  segmente distincte, reuniunea cărora este  $[a, b]$ . Pentru simplitate, punctele diviziunii vor fi echidistante (fig. 3.10). Atunci lungimea  $h$  a fiecărui segment elementar  $[x_i, x_{i+1}]$  va fi determinată de formula

$$h = \frac{|b-a|}{n}.$$

Valorile  $x_i$  pot fi și ele exprimate prin mărimi cunoscute:  $x_i = a + ih$ ,  $i = 0, \dots, n$ .

Fiind cunoscute valorile  $x_i$  și  $x_{i+1}$ , poate fi determinat și mijlocul  $z_i$  al fiecărui segment elementar  $z_i = \frac{x_i + x_{i+1}}{2}$ .

Pe fiecare dintre segmentele  $[x_i, x_{i+1}]$  aria trapezului curbiliniu va fi aproximată prin aria dreptunghiului  $S_i$ , având lungimile laturilor  $h$  și  $f(z_i)$  (fig. 3.11):

$$S_i = hf(z_i) = hf(a + ih + \frac{h}{2}).$$

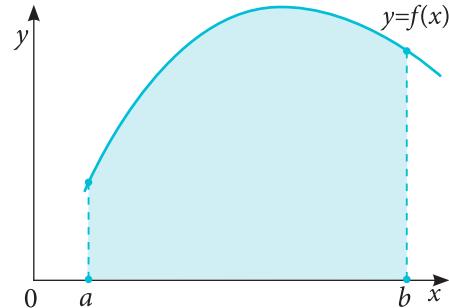


Fig. 3.9. Sensul geometric al integralei definite

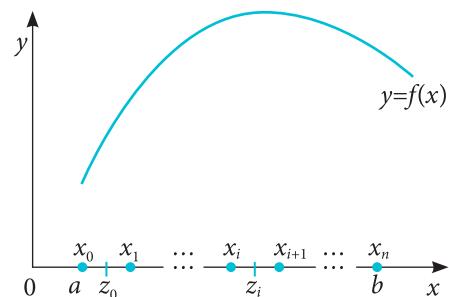


Fig. 3.10. Divizarea  $[a, b]$  în segmente elementare. Mijlocul segmentului elementar  $[x_i, x_{i+1}]$  este punctul

$$z_i = \frac{x_i + x_{i+1}}{2}$$

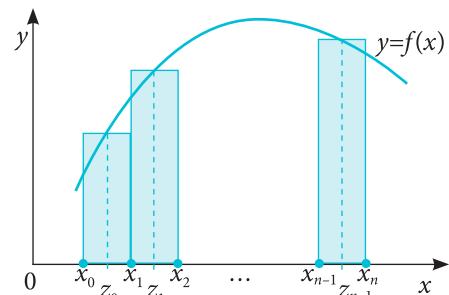


Fig. 3.11. Construcția consecutivă a dreptunghiurilor, care aproximează figura inițială. Pe fiecare segment elementar înălțimea dreptunghiului este determinată de valoarea funcției  $f(x)$  în mijlocul segmentului

În acest caz, valoarea calculată  $I$  a integralei definite se calculează ca suma ariilor dreptunghiurilor și se determină nemijlocit din formula

$$I = h \sum_{i=0}^{n-1} f\left(a + ih + \frac{h}{2}\right),$$

unde

$n$  – numărul de divizări ale segmentului inițial;

$h$  – lungimea segmentului elementar;

$I$  – valoarea calculată a integralei.

Astfel, calculul integralei se transformă în calculul valorii unei expresii aritmetice, care depinde doar de numărul de diviziuni ale segmentului de integrare și de valoarea funcției în punctele de mijloc ale segmentelor elementare. Metoda care reduce calculul integralei la calculul unei sume de arii a dreptunghiurilor este numită *metoda dreptunghiurilor*.

### Eroarea metodei

Valoarea calculată a integralei este de cele mai multe ori diferită de valoarea exactă, calculată analitic. Eroarea apare din motivul aproximării pe fiecare segment elementar a funcției  $f(x)$  cu o funcție constantă  $g$ , iar mărimea erorii  $\varepsilon$  este determinată de integrala erorii de aproximare și poate fi calculată după formula:

$$\varepsilon = \left| \int_a^b f(x) dx - I \right| \leq (b-a)M \frac{h}{4},$$

unde  $M$  – supremul  $|f'(x)|$  pe  $[a, b]$ ,  $I$  – valoarea calculată a integralei.

Din formula precedentă rezultă un fapt important: eroarea de calcul a metodei dreptunghiurilor este proporțională cu numărul de divizări ale segmentului de integrare. Astfel, pentru a obține o eroare de calcul, ce nu depășește o valoare prestatibilită  $\varepsilon$ , este suficient să se realizeze divizarea segmentului de integrare în  $n$  segmente elementare, valoarea lui  $n$  fiind determinată prin secvența de transformări

$$(b-a)M \frac{h}{4} \leq \varepsilon \Rightarrow \frac{(b-a)^2 M}{4n} \leq \varepsilon \Rightarrow n = \left\lceil \frac{(b-a)^2 M}{4\varepsilon} \right\rceil + 1.$$

### Algoritmizarea metodei

Deoarece în cazul calculelor cu o eroare ce nu depășește valoarea prestatibilită  $\varepsilon$  numărul necesar de divizări poate fi stabilit apriori, este suficient să se realizeze un singur algoritm – pentru un număr fixat de divizări  $n$ :

**Pasul 1. Inițializare:** Se introduc valorile extremităților segmentului de integrare  $a$ ,  $b$  și numărul de divizări  $n$ .

**Notă.** În cazul în care este necesar calculul integralei cu o eroare ce nu depășește o valoare prestatibilită  $\varepsilon$ , numărul de divizări  $n$  se calculează cu ajutorul formulei  $n = \left\lceil \frac{(b-a)^2 M}{4\varepsilon} \right\rceil + 1$ .

**Pasul 2.** Se calculează lungimea segmentului elementar  $h = \frac{|b-a|}{n}$ .  $S \leftarrow 0$ .

**Pasul 3.** Pentru toți  $i$  de la 0 la  $n-1$ :  $\frac{h}{2}$

a) Se calculează valorile  $z_i \leftarrow a + ih + \frac{h}{2}$ .

b) Se calculează aria dreptunghiului elementar:  $S_i \leftarrow f(z_i) \times h$ .

c) Aria calculată se sumează cu ariile precedente:  $S \leftarrow S + S_i$ .

**Pasul 4.** Se afișează aria totală calculată  $S$ . SFÂRȘIT.

**Exemplul 1:** Să se scrie un program care calculează integrala  $\int_1^2 \frac{x^4}{\sqrt{1+x}} dx$  prin 20, 40, 80 și 160

de divizări ale segmentului de integrare, folosind metoda dreptunghiurilor. Pentru fiecare număr de divizări se va indica valoarea calculată cu șase semne după virgulă.

**Rezolvare:** Deoarece numărul de divizări este indicat în condițiile problemei, preprocesarea matematică nu este necesară.

```
program cn09;
const r=4;
var S, a, b, h : real;
j,i,n:integer;
function f(x:real): real;
begin
f:=x*x*x*x/sqrt(1+x);
end;
begin
a:=1; b:=2; n:=10;
for j:=1 to r do
begin
S:=0; n:=n*2; h:=(b-a)/n;
for i:=0 to n-1 do
S:=S+ h*f(a + i*h + h/2);
writeln (' n=',n,' I=',S:0:6);
end;
end.
```

Rezultate:    n=20 I=3.788513  
n=40 I=3.789629  
n=80 I=3.789908  
n=160 I=3.789977

```
// program cn09;
#include <iostream>
#include <iomanip>
#include <math.h>
using namespace std;
#define r 4

double S, a, b, h;
int j, i, n;

double f(double x)
{
    return pow(x, 4) / sqrt(1 + x);
}

int main()
{
    a = 1; b = 2; n = 10;
```

```

for (j = 1; j <= 4; j++)
{
    S = 0; n = n * 2;
    h = (b - a) / n;
    for (i = 0; i < n; i++)
        S = S + h * f(a + i * h + h / 2);
    cout<< setprecision(8)<< "n = " << n << " I=" << S << endl;
}
return 0;
}

```

*Rezultate:*

n = 20	I=3.7885128
n = 40	I=3.7896286
n = 80	I=3.7899076
n = 160	I=3.7899774

**Exemplul 2:** Să se scrie un program care calculează valoarea aproximativă a integralei  $\int_{-\frac{\pi}{2}}^{\frac{\pi}{2}} \frac{\sin(\cos x^2)}{2} dx$  prin metoda dreptunghiurilor cu o eroare ce nu depășește valoarea dată  $\varepsilon$ . Calculul va fi oprit, dacă se îndeplinește condiția:  $\frac{\pi}{8}h \leq \varepsilon$ ;  $h = \frac{|b-a|}{n}$ , unde  $n$  – numărul curent de divizări.

Valorile  $a, b, \varepsilon$  se definesc nemijlocit în program.

**Rezolvare:** Din condițiile problemei se deduce numărul necesar de divizări:

$$\frac{\pi|b-a|}{8n} \leq \varepsilon; \Rightarrow n = \left\lceil \frac{\pi|b-a|}{8\varepsilon} \right\rceil + 1.$$

```

program* cn10;
var S, a, b, e, h : real;
j,i,n:integer;
function f(x:real): real;
begin f:=sin(cos (x*x))/2;end;
begin a:=-pi/2; b:=pi/2; e:=0.0001;
n:=round(pi*(abs(b-a))/(8*e))+1;
S:=0;
h:=(b-a)/n;
for i:=0 to n-1 do
S:=S+ h*f(a + i*h + h/2);
writeln ('n=',n,' I=',S:0:6);
end.

```

*Rezultate:* n=12338 I=0.729729

---

\* Aici și în alte programe din prezenta ediție se folosește constanta predefinită pi, utilizată în mediul Turbo Pascal. Utilizatorii altor limbi de programare sau ai altor compilatoare Pascal vor defini constanta respectivă în cadrul programului. Ex.: const pi=3.141.

```
// program cn10;
#include <iostream>
#include <iomanip>
#include <math.h>
    using namespace std;
#define pi 3.14159265359

double S, a, b, e, h;
int i, n;

double f(double x)
{
    return sin(cos (x * x)) / 2;
}

int main()
{
    a = -pi / 2; b = pi / 2; e = 0.0001;
    n = round(pi * (abs(b - a))/(8 * e)) + 1;
    S = 0;
    h = (b - a) / n;
    for (i = 0; i < n; i++)
        S = S + h * f(a + i * h + h / 2);
    cout << setprecision(8) << "n = " << n << " I=" << S;
    return 0;
}
```

Rezultat: n = 12338 I=0.72972935

## Întrebări și exerciții

- 1 Argumentează!** Motivați necesitatea calculului numeric al integralei definite.
- 2 Explică!** Ce procedeu stă la baza metodei dreptunghiurilor pentru calculul numeric al integralei?
- 3 Explică!** Care este relația dintre numărul de divizări ale segmentului de integrare și eroarea de calcul a metodei dreptunghiurilor?
- 4 Cercetează!** Elaborați un program care calculează integrala definită prin metoda dreptunghiurilor, pentru 200, 2000, 20 000 de divizări ale segmentului de integrare. Pentru fiecare număr de divizări valoarea calculată se va afișa cu șase cifre zecimale.

a)  $\int_{-1}^1 \frac{1+x^2}{1+x^4} dx;$

c)  $\int_0^{2\pi} \frac{dx}{(2+\cos x)(3+\cos x)};$

e)  $\int_0^1 \frac{dx}{(x+1)\sqrt{x^2+1}};$

b)  $\int_1^9 \left(x^3\sqrt[3]{1-x}\right) dx;$

d)  $\int_0^{2\pi} \frac{dx}{\sin^4 x + \cos^4 x};$

f)  $\int_0^{\ln 2} \sqrt{e^x - 1} dx.$

Explicați diferența dintre rezultatele obținute pentru valori diferite ale numărului de divizări.

Verificați rezultatele obținute folosind calculatorul online pentru integrale definite: [www.solvemymath.com/online\\_math\\_calculator/calculus/definite\\_integral](http://www.solvemymath.com/online_math_calculator/calculus/definite_integral).

- 5** **Integreză!** Scrieți un program care calculează integrala definită prin metoda dreptunghiurilor cu o eroare care nu va depăși valoarea prestabilită  $\varepsilon = 0,001; \varepsilon = 0,0001; \varepsilon = 0,00001$ . Valoarea  $M$  este cunoscută apriori:

a)  $\int_0^3 (x^2 + 3x) dx,$

$M = 9;$

c)  $\int_1^2 (x^3 - 7x) dx,$

$M = 5;$

b)  $\int_1^2 (x^3 - 2x^2 + 3x) dx,$

$M = 7;$

d)  $\int_2^7 (x^2 \sqrt{x-1}) dx,$

$M = 36.$

Pentru fiecare valoare a erorii se va afișa valoarea calculată a integralei și numărul de divizări pentru care se obține aceasta.

- 6** **Integreză!** Conturul unui teren este determinat lateral de dreptele verticale  $x = -2$  și  $x = 0$ , superior – de graficul funcției  $f(x) = x^3 - 5x + 3 \cos x$ , inferior – de graficul funcției  $f(x) = x^3 - x^2$ . Scrieți un program care va calcula aria terenului, folosind metoda dreptunghiurilor, pentru:

- a) 10 000 de divizări;      b)  $\varepsilon = 0,00001$ .

### 3.11 Variații ale metodei dreptunghiurilor

În unele cazuri, punctul care determină înălțimea dreptunghiului ce aproximează integrala pe un segment elementar este selectat nefiind mijlocul segmentului elementar, ci una dintre extremitățile lui. În acest caz, dacă aproximarea este realizată prin extremitățile stângi (fig. 3.12) ale segmentelor elementare, aria dreptunghiului elementar  $S_i = hf(x_i)$ , iar integrala definită se aproximează prin suma:

$$I_{st} = \sum_{i=0}^{n-1} hf(x_i) = h \sum_{i=0}^{n-1} f(a + ih),$$

unde  $h = \frac{|b-a|}{n}$ ,  $n$  – numărul de divizări ale segmentului inițial, iar metoda este numită *metoda dreptunghiurilor de stânga*.

Dacă aproximarea este realizată prin extremitățile drepte (fig. 3.13) ale segmentelor elementare, aria dreptunghiului elementar  $S_i = hf(x_{i+1})$ , iar integrala definită se aproximează prin suma:

$$I_{dr} = \sum_{i=1}^n hf(x_i) = h \sum_{i=1}^n f(a + ih),$$

unde  $h = \frac{|b-a|}{n}$ ,  $n$  – numărul de divizări ale segmentului inițial, iar metoda este numită *metoda dreptunghiurilor de dreapta*.

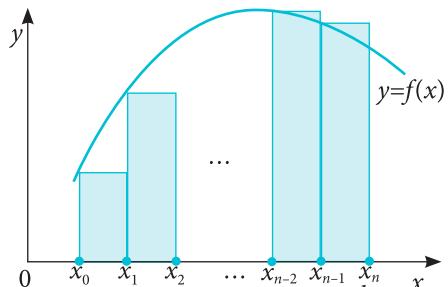


Fig. 3.12. Aproximarea integrală definită prin dreptunghiuri de stânga

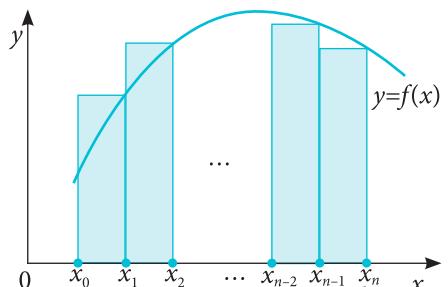


Fig. 3.13. Aproximarea integrală definită prin dreptunghiuri de dreapta

Eroarea de calcul pentru variațiile metodei dreptunghiurilor.

În cazul aproximării integralei definite prin arii ale dreptunghiurilor de stânga (dreapta), formula de estimare a erorii se modifică nesemnificativ în raport cu formula de bază:

$$\left| \int_a^b f(x)dx - I \right| \leq (b-a) M \frac{h}{2},$$

unde  $M$  – supremul  $|f'(x)|$  pe  $[a, b]$ ,  $I$  – valoarea calculată a integralei prin dreptunghiuri de stânga (dreapta).

La fel ca în cazul dreptunghiurilor de mijloc, numărul de divizări necesare pentru a obține o soluție calculată cu o eroare ce nu depășește o valoare prestabilită  $\varepsilon$  poate fi dedus direct din formula erorii:

$$n = \left\lceil M \frac{(b-a)^2}{2\varepsilon} \right\rceil + 1.$$

### Algoritmizarea metodei (dreptunghiuri de stânga)\*

**Pasul 1.** *Inițializare:* Se introduc valorile extremităților segmentului de integrare  $a, b$  și numărul de divizări  $n$ .

*Notă.* În cazul în care este necesar calculul integralei cu o eroare care nu depășește o valoare prestabilită  $\varepsilon$ , numărul de divizări  $n$  se calculează cu ajutorul formulei  $n = \left\lceil \frac{(b-a)^2 M}{2\varepsilon} \right\rceil + 1$ .

**Pasul 2.** Se calculează lungimea segmentului elementar  $h = \frac{|b-a|}{n}$ .  $S \Leftarrow 0$ .

**Pasul 3.** Pentru toți  $i$  de la 0 la  $n-1$ :

- Se calculează valorile  $x_i \Leftarrow a + ih$ .
- Se calculează aria dreptunghiului elementar:  $S_i \Leftarrow f(x_i) \times h$ .
- Aria calculată se sumează cu ariile precedente:  $S \Leftarrow S + S_i$ .

**Pasul 4.** Se afișează aria totală calculată  $S$ . SFÂRȘIT.

**Exemplul 3:** Să se calculeze integrala definită  $\int_0^\pi (x \sin(x))^2 dx$ , utilizând metoda dreptunghiurilor, variația dreptunghiurilor de stânga, pentru 10, 100, 1 000 de divizări. Atribuirea valorilor inițiale se face direct în program. Pentru fiecare număr de divizări la ecran se va afișa valoarea calculată și numărul de divizări, separate prin spațiu.

```
program cn11;
const r=3;
var S, a, b, h : real;
j,i,n:integer;
function f(x:real): real;
begin      f:=sqr((x* sin(x)));end;
begin      a:=0; b:=pi; n:=1;
for j:=1 to r do
begin S:=0; n:=n*10;
h:=(b-a)/n;
```

\* Pentru dreptunghiurile de dreapta, algoritmul se deosebește doar în pasul 3 prin diapazonul valorilor pe care le primește indicele  $i$ : de la 1 la  $n$ .

```

for i:=0 to n-1 do
  S:=S+ h*f(a + i*h);
  writeln ('n=',n,' I=',S:0:6);
  end;
end.

```

*Resultate:*

```

n=10 I=4.381796
n=100 I=4.382315
n=1000 I=4.382315

```

```

// program cn11;
#include <iostream>
#include <iomanip>
#include <math.h>
using namespace std;

#define r 3
#define pi 3.14159265359

double S, a, b, h;
int j, i, n;

double f(double x)
{
    return pow(x * sin(x), 2);
}

int main()
{
    a = 0; b = pi; n = 1;
    for (j = 1; j <= r; j++)
    {
        S = 0; n = n * 10;
        h = (b - a) / n;
        for (i = 0; i < n; i++)
            S = S + h * f(a + i * h);
        cout<< setprecision(8)<< "n = " << n << " I=" << S << endl;
    }
    return 0;
}

```

*Resultate:*

```

n = 10 I=4.3827687
n = 100 I=4.3823147
n = 1000 I=4.3823146

```

**Exemplul 4:** Să se calculeze integrala definită  $\int_0^1 \sin x \sin 2x \sin 3x dx$ , utilizând metoda dreptunghiurilor, variația dreptunghiurilor de dreapta, cu o eroare de calcul ce nu depășește valoarea  $\varepsilon = 0,001$ . Atribuirea valorilor inițiale se face direct în program. La ecran se va afișa valoarea calculată a integralei și numărul de divizări necesare pentru a obține precizia cerută, separate prin spațiu.

**Rezolvare:** Numărul necesar de divizări poate fi determinat analitic.

Prin calcule elementare se determină valoarea  $M = 6$ :

$$n = \left\lceil M \frac{(b-a)^2}{2\varepsilon} \right\rceil + 1.$$

```
program cn12;
var S, a, b, e, h, M : real;
j,i,n:integer;
function f(x:real): real;
begin    f:=sin(x)* sin(2*x)* sin(3*x);end;
begin    a:=0; b:=1; e:=0.001; M:=6;
         n:=trunc(M*(b-a)*(b-a)/(2*e))+ 1;
         S:=0; h:=(b-a)/n;
         for i:=1 to n do
           S:=S+ h*f(a + i*h);
         writeln ('n=',n,' I=',S:0:6);
end.
```

Rezultate: n=3001 I=0.278729

```
// program cn12;
#include <iostream>
#include <math.h>
using namespace std;

double S, a, b, e, h, M;
int j, i, n;

double f(double x)
{
    return sin(x) * sin(2 * x) * sin (3 * x);
}

int main()
{
    a = 0; b = 1; e = 0.001; M = 6;
    n = trunc(M * pow((b-a),2) / (2 * e)) + 1;
    S = 0;
    h = (b - a) / n;
```

```

for (i = 1; i <= n; i++)
    S = S + h * f(a + i * h);
cout<< "n = " << n << " I=" << S << endl;
return 0;
}

```

*Rezultate: n = 3001 I=0.278729*

## Întrebări și exerciții

- 1 **Explică!** Care este diferența dintre metoda dreptunghiurilor de mijloc și variațiile sale?
- 2 **Explică!** Care este relația dintre numărul de divizări ale segmentului de integrare și eroarea de calcul a variațiilor metodei dreptunghiurilor?
- 3 **Analizează!** Care dintre variațiile metodei dreptunghiurilor este mai exactă? De ce?
- 4 **Cercetează!** Elaborați un program care calculează integrala definită prin metoda dreptunghiurilor de stânga, de mijloc, de dreapta. Utilizați un număr variabil de divizări (de exemplu 10, 100, 1 000).

a)  $\int_1^3 (x \ln x)^2 dx;$

b)  $\int_1^6 (x\sqrt{x-1}) dx;$

c)  $\int_0^1 (x^2 \sqrt{1+3x^2}) dx;$

d)  $\int_0^{\pi} (e^x \cos^2 x) dx;$

e)  $\int_0^{\pi} (x \sin x)^2 dx;$

f)  $\int_0^{2\pi} \frac{dx}{(2+\cos x)(3+\sin x)}.$

Explicați diferența dintre rezultatele obținute pentru valori diferite ale numărului de divizări.

Verificați rezultatele obținute folosind calculatorul online pentru integrale definite: [www.solvemymath.com/online\\_math\\_calculator/calculus/definite\\_integral](http://www.solvemymath.com/online_math_calculator/calculus/definite_integral).

- 5 **Programează!** Elaborați un program care calculează integrala definită prin metoda dreptunghiurilor cu o eroare care nu va depăși valoarea prestabilită  $\varepsilon = 0,005$ ;  $\varepsilon = 0,0005$ ;  $\varepsilon = 0,00005$ . Valoarea  $M$  urmează să fie determinată prin metode analitice:

a)  $\int_1^3 (3x^2 + x + 2) dx;$     b)  $\int_1^2 (2x^3 - x^2 + 5x) dx;$     c)  $\int_{-3}^{-1} (2x^3 - 3x) dx.$

Pentru fiecare valoare a erorii se va afișa valoarea calculată a integralei și numărul de divizări, pentru care se obține aceasta.

- 6 **Integreză!** Elaborați un program ce va determina, folosind variațiile metodei dreptunghiurilor, care dintre integralele definite de mai jos este mai mare:

a)  $\int_0^{\pi} (e^{-x^2} \cos^2 x) dx$  sau  $\int_{\pi}^{2\pi} (e^{-x^2} \cos^2 x) dx;$     b)  $\int_0^1 (e^{-x}) dx$  sau  $\int_0^1 (e^{-x^2}) dx.$

# Baze de date

## 4.1 Date și baze de date

### Date elementare și structuri de date

**Data** este un model de reprezentare a informației, accesibilă unui procesor (om, program etc.), model cu care se poate opera pentru a obține informații noi.

**Data elementară** este o entitate indivizibilă atât în raport cu informația pe care o reprezintă, cât și în raport cu procesorul logic (programul) sau cel fizic (unitatea centrală care o prelucrează).

Din punct de vedere logic, data elementară este caracterizată de:

- a) *identificatorul datei*;
- b) *valoarea datei*;
- c) *atributele datei*.

**Identificatorul datei** se folosește pentru a apela (a accesa) această dată.

**Valoarea datei** este conținutul zonei de memorie în care este stocată data. **Domeniul de definiție al datei** este mulțimea valorilor pe care le poate lua data în procesul prelucrării ei.

**Atributele datei** stabilesc caracteristici specifice datei. Unul dintre cele mai importante atrbute ale datei este **tipul**, care definește apartenența datei la o anumită clasă de date. Fiecare tip de date îi corespunde un model de reprezentare internă.

**Exemplul 1:** Declarația var c: integer, din Pascal, definește variabila c care în orice moment de timp al execuției programului identifică o dată elementară de tip *integer* ce va „ocupa” o zonă de memorie de doi octeți. Domeniul de definiție al tipului *integer* este mulțimea numerelor  $\{-32\ 768, -32\ 767, \dots, 32\ 767\}$ . În urma instrucțiunii `c := 8`, în zona de memorie menționată se va stoca valoarea 8.

**Exemplul 2:** Valorile tipurilor simple din Pascal sunt date elementare.

**Structura de date** este un ansamblu de date, între elementele căruia sunt stabilite anumite relații care definesc metodele de identificare și prelucrare a acestor date. Structura de date poate fi compusă din date elementare sau din alte structuri de date. Fiecare componentă a structurii de date poate fi localizată cu ajutorul identificatorului ei sau prin poziția ei în cadrul structurii.

**Exemplul 3:** Valorile tipului *record* din Pascal sunt structuri de date ale căror componente (câmpurile) pot fi apelate folosind numele lor, adică identificatorii câmpurilor. Un masiv este o structură de date, ale cărui componente (elementele masivului) pot fi apelate utilizând poziția acestei componente (adică indicele ei).

Principalele caracteristici ale unei structuri de date sunt:

- Omogenitatea.* O structură *omogenă* are toate componentele de același tip. Astfel, în-registrările (valori de tip *record*) sunt structuri neomogene, iar masivele – structuri omogene;
- Modul de acces al componentelor.* Componentele unei structuri pot fi accesate direct sau secvențial. De exemplu, fișierele text sunt structuri de date cu acces secvențial;
- Stabilitatea structurii.* Dacă pe parcursul execuției programului o structură nu-și schimbă numărul de elemente și relațiile dintre ele, atunci această structură este *statică*. În caz contrar, o astfel de structură se numește *structură dinamică*. De exemplu, listele înlănțuite sunt structuri dinamice, iar înregistrările – structuri statice;
- Timpul de utilizare.* Structurile pot fi permanente sau temporare. De exemplu, fișierele externe sunt structuri de date permanente, iar masivele – structuri temporare.

## Baze de date

Până în prezent am prelucrat diferite tipuri de date și structuri de date utilizând un limbaj de programare sau o aplicație *software* (editor de texte, procesor tabelar etc.). Cea mai complexă structură de date prelucrată a fost fișierul.

Evident, informațiile despre o firmă, companie sau instituție pot fi păstrate în fișiere. De exemplu, în cazul unui liceu, se pot crea fișierele *Profesori*, *Elevi*, *Clase*, *Discipline* etc., în care se vor stoca datele, respectiv, despre profesori, elevi, clase, discipline studiate etc. Pentru a păstra informații despre repartizările profesorilor pe clase, se va crea alt fișier, *Profesor\_Clasă*, pe baza conținuturilor fișierelor *Profesori* și *Clase*. Să presupunem că un șef de studii proaspăt angajat creează acest fișier. Dacă informația despre un profesor (din fișierul *Profesori*) este greșită, atunci ea va apărea eronată și în tabelul *Profesor\_Clasă*. Eventual, aceeași greșală va fi și în fișierul *Orarul\_lecțiilor*. Sesizând-o, șeful de studii va fi nevoit să corecteze pe rând toate cele trei fișiere. Mult mai simplu ar fi dacă, actualizând fișierul *Profesori*, automat vor fi actualizate și toate fișierele care „l-au utilizat”. În acest mod, s-ar asigura **integritatea datelor**.

În afară de *complexitatea actualizărilor*, metoda prelucrării datelor prin fișiere independente are și alte dezavantaje:

- *Prezența acelorași date în câteva fișiere independente* poate duce la duplicarea datelor, deci și la consumul mare de memorie.
- *Dificultatea obținerii informațiilor spontane* rezidă din independența fișierelor (de exemplu, pentru a obține lista profesorilor de sex feminin care predau științele exacte în clasa a X-a, profil real, trebuie prelucrate cel puțin fișierele *Profesori*, *Clase*, *Discipline*).
- *Formatele incompatibile de fișiere* încetinesc prelucrarea informațiilor (de exemplu, fișierele generate cu diferite limbaje de programare este posibil să aibă formate diferite, deci va fi necesară o aplicație care le va transforma în fișiere cu același format).

Acestea și alte probleme pot fi rezolvate dacă se folosește o structură de date de tip *bază de date*.

O **bază de date** este un ansamblu de informații (date), organizate într-un mod special, fapt care facilitează stocarea și extragerea lor. Informațiile dintr-o bază de date sunt păstrate sub formă de înregistrări sau de fișiere, între care există legături logice. În afară de depozitul de informații, baza de date conține descrierea lor: tipul, structura, modul de organizare și relațiile dintre ele.

## Întrebări și exerciții

**1 Sistematizează-ți cunoștințele!** Numiți tipurile de date studiate. Care dintre ele sunt:

- a) tipuri de date elementare;
- b) tipuri de date structurate?

**2 Exersează!** Selectați datele elementare: simbol, număr real, sir de caractere, masiv de numere, element al unei mulțimi, număr întreg, valoarea *false*, fișier.

**3 Explică!** Care este domeniul de definiție al tipului:

- a) integer;
- b) char;
- c) lit\_M declarat astfel `type lit_M = 'A'.. 'z'`;
- d) boolean?

**4 Analizează!** Caracterizați structurile de date:

- a) mulțime;
- b) masiv;
- c) înregistrare;
- d) listă înlănțuită;
- e) fișier.

Repere: omogenitate, mod de acces, stabilitate și timp de utilizare.

**5 Explicați sensul noțiunii bază de date.**

**6 Analizează!** Argumentați de ce pentru păstrarea și prelucrarea informației unei companii este mai avantajos utilizarea unei baze de date comparativ cu utilizarea unui sistem de fișiere.

**7 Explică!** Care sunt deosebirile dintre o bază de date și o structură de date?

### 4.2 Tipuri de baze de date

După cum vom vedea în capitolul următor, un pas important în elaborarea bazelor de date este proiectarea *modelului logic* de date al bazei, care nu depinde de parametrii fizici ai mediului de păstrare a datelor și definește modul lor de organizare.

**Modelul logic al unei baze de date reprezintă o descriere generală a bazei de date cu ajutorul limbajului natural, al celui matematic, al diagramelor și organigramelor, al graficelor și al altor mijloace înțelese de cei care elaborează baza de date.**

Cele mai răspândite modele logice sunt:

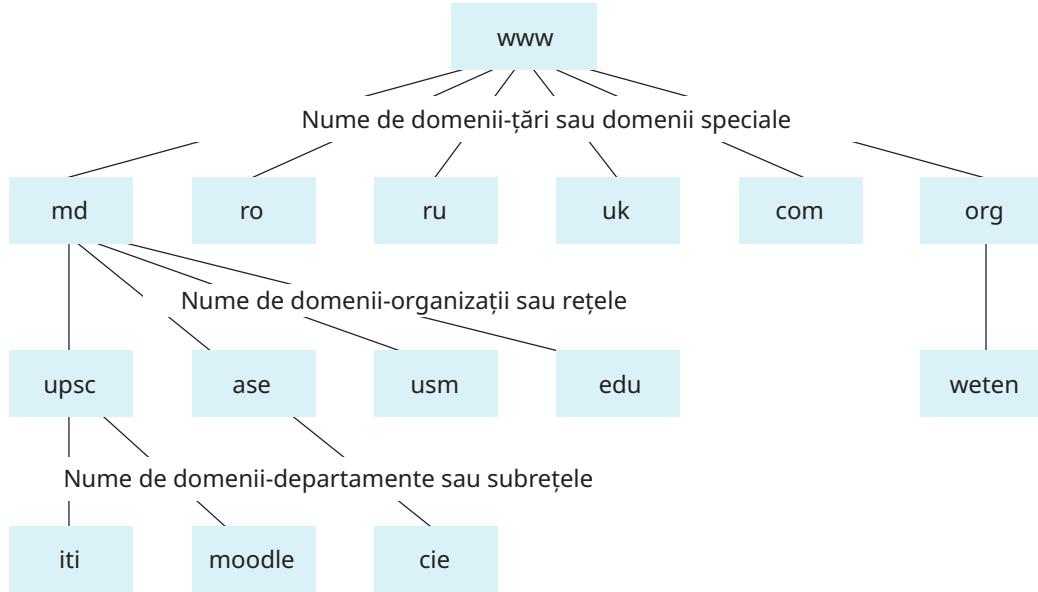
- a) *ierarhic* (sau *arborescent*);
- b) *rețea*;
- c) *relațional*.

În funcție de modelul ei logic, baza de date se numește *ierarhică*, *în rețea* sau *relațională*.

Entitățile (colecțiile de date) ale **bazei de date ierarhice** (*hierarchical database*) sunt organizate sub formă de *noduri*, la care sunt conectate ramurile unui *arbore*. Fiecare nod se subordonează

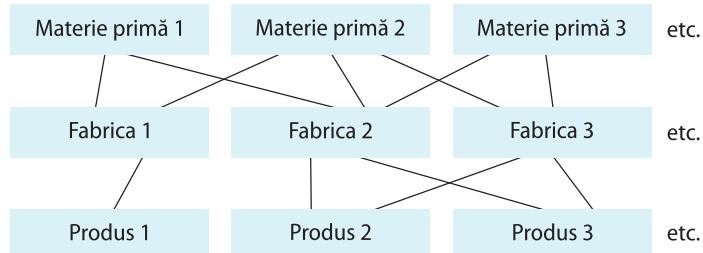
cel mult unui nod de nivel ierarhic imediat superior (numit *nod-părinte*) și poate fi legat cu nici-unul, unul sau cu câteva noduri de nivel inferior (numite *noduri-copii*).

**Exemplu:** Baza de date a adreselor Internet are următorul model ierarhic:



În cazul **bazei de date în rețea** (*network database*), entitățile la fel sunt aranjate ierarhic, însă unui nod-copil poate să-i corespundă câteva noduri-părinte.

**Exemplu:** Baza de date a unei ramuri industriale poate avea următorul model:



**O bază de date relatională** (*relational database*) are cel mai flexibil model conceptual de organizare a datelor. Parcurgerea entităților unei astfel de baze nu este ierarhică.

Modelul relațional este simplu, dar riguros din punct de vedere matematic. Pentru prima dată acest model a fost propus în 1970 de savantul Edgar Frank Codd\*.

Baza de date relatională este formată din tabele, denumite *relații*. Fiecare tabel este format din rânduri și coloane. Rândurile se numesc *înregistrări* de date, iar coloanele – *câmpuri* de date. *Capul de tabel* (sau *antetul tabelului*) definește structura tabelului. Prin *crearea unui tabel* se subînțelege, de fapt, definirea capului de tabel.

\* Edgar Frank Codd (23.08.1923, Insula Portland, Anglia – 18.04.2003, Williams Island, Florida, SUA) – informatician american de origine engleză care, lucrând pentru IBM, a inventat modelul relațional pentru gestiunea bazelor de date. Pentru contribuțiile sale în domeniul informaticii, în 1981 obține Premiul Turing, considerat „Premiul Nobel pentru Informatică”.

În figura 4.1 este reprezentată o parte din schema unei baze de date relaționale.

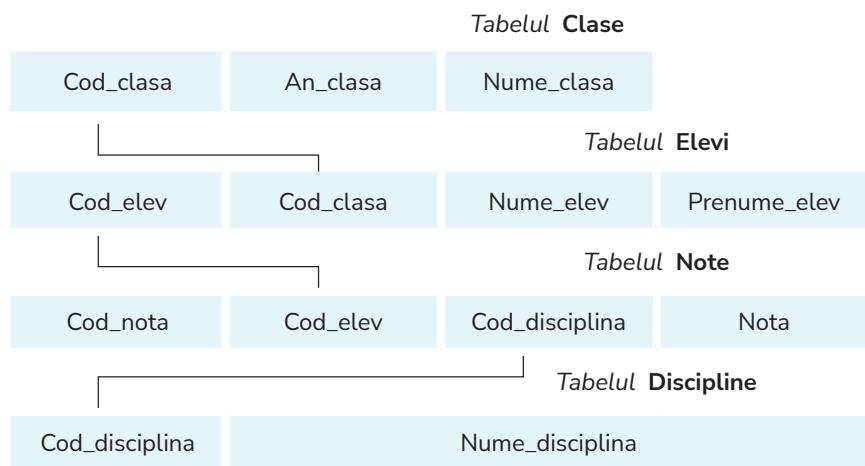


Fig. 4.1. Parte dintr-o schemă a unei baze de date relaționale

Între tabelele unei baze de date relaționale există interdependențe (despre care vom afla mai detaliat în următoarele paragrafe).

**Observație:** În continuare vom examina doar baze de date relaționale.

## Întrebări și exerciții

- 1 Sistematizează-ți cunoștințele!** Descrieți structura bazelor de date:
  - a) ierarhice;
  - b) în rețea;
  - c) relaționale.
- 2 Aplică!** Stabiliți tipul bazei de date știind că:
  - a) între fiecare două entități ale bazei există o legătură;
  - b) entitățile bazei sunt noduri ale unui arbore binar;
  - c) o entitate este legată cu toate celelalte entități și alte legături nu există;
  - d) informațiile din bază sunt organizate în zece tabele.
- 3 Elaborează!** Reprezentați schematic un model ierarhic pentru o bază de date cu informații despre:
  - a) un liceu;
  - b) un magazin de produse alimentare;
  - c) o bibliotecă.
- 4 Învață să înveți!** Rezolvați exercițiul 3 pentru cazul:
  - a) modelul rețea;
  - b) modelul relațional.
- 5 Aplică!** Stabiliți tipul bazei de date a numerelor de telefoane fixe din țară.

4.3

## Elaborarea bazelor relaționale de date

### Aspecte generale

Un **sistem de bază de date** reprezintă un sistem de organizare și de prelucrare a bazei de date și este format din baza de date propriu-zisă și dintr-un ansamblu de programe prin care se asigură gestionarea și prelucrarea complexă a datelor.

Prin **proiectarea** sau **elaborarea unei baze de date** se subînțelege crearea sistemului acestui baze de date.

Proiectarea unei baze de date constă, de regulă, din următoarele **etape**:

- Etapa de analiză.** Se analizează domeniul de date și se depistează cerințele tuturor categoriilor de utilizatori către sistemul de bază de date, precum și operațiile pe care le va gestiona acest sistem. În urma acestei analize se proiectează modelul conceptual al bazei de date. De asemenea, se stabilește structura interfeței principale a sistemului de bază de date și a celor auxiliare.
- Etapa de programare.** Se creează componentele logice (programele): interfață principală, aplicațiile de introducere/actualizare/prelucrare a datelor, cele de interogare a bazei de date și de extragere a informațiilor.
- Punerea în funcțiune și exploatarea sistemului de bază de date.** Baza de date se completează cu date (înregistrări). În continuare se realizează operații de actualizare, de consultare și de întreținere ulterioară (inclusiv dezvoltare) a sistemului de bază de date.

Elaborarea și întreținerea unei baze de date se realizează, în genere, de un grup de persoane: *administratori* (care stabilesc structura inițială a bazei de date, modul de memorare a datelor la nivel fizic; acordă utilizatorilor drepturi de acces la baza de date, asigură securitatea datelor, modifică structura și dezvoltă baza de date), *programatori* (care scriu programele în limbaje de programare), *proiectanți* (care realizează designul interfețelor bazei de date) etc.

**Observație:** Există sisteme computerizate moderne, numite Sisteme de Gestire a Bazelor de Date (SGBD), care au fost elaborate special pentru a crea sisteme de bază de date. Mai detaliate aceste SGBD-uri vor fi examinate în paragraful următor.

### Proiectarea entităților unei baze de date relaționale

În paragraful precedent am menționat că entitățile unei baze de date relaționale sunt **tabelele** în care se păstrează **înregistrări** de date (rânduri de date).

*Cuprinsul tabel* (sau *antetul tabelului*) definește structura tabelului. Prin *crearea unui tabel* se subînțelege, de fapt, definirea capului de tabel, adică descrierea câmpurilor (a coloanelor) tabelului.

**Exemplu:** Considerăm o bază de date, denumită *Liceu*, în care sunt stocate informații despre un liceu. Tabelul *Elevi* al acestei baze de date păstrează informații despre elevii liceului:

Tabelul **Elevi**

Cod_elev	Nume_elev	Pren_elev	Cod_clasa	Data_elev	Foto_elev	Telefon	Gen_elev
e001	Bacinschi	Sabina	c01	28.09.2007	Package	29-82-54	F
e002	Belobrov	Andreea	c01	18.10.2007	Package	44-26-48	F
e006	Chilimciuc	Dumitru	c01	09.08.2007	Package	26-13-16	M

La descrierea fiecărui câmp al unui tabel se specifică *numele* și *tipul* câmpului.

- a) *Numele câmpului* trebuie să fie unic în cadrul tabelului. Datele unui câmp sunt omogene;
- b) *Tipul câmpului* definește tipul valorilor câmpului (numeric, sir de caractere, text lung, dată calendaristică, logic, imagine etc.).

Astfel, primul câmp are numele *Cod\_elev* și tipul *Sir de caractere*, iar al cincilea câmp – numele *Data\_elev* și tipul *Dată calendaristică*.

Fiecare valoare din câmpul *Cod\_elev* al tabelului *Elevi* corespunde unei singure înregistrări.

Câmpul ale căruia valori identifică fiecare înregistrare a tabelului se numește **cheie primară**. Evident, un câmp este cheie primară doar dacă valorile lui nu sunt nule și nu se repetă.

Se recomandă ca fiecare tabel al unei baze de date să conțină un câmp cheie primară. Uneori, rolul cheii primare îi poate reveni câtorva câmpuri (dacă acestea împreună identifică unic rândurile tabelului).

Un câmp al unui tabel se numește **cheie externă** dacă el este definit drept **cheie primară** în alt tabel. Astfel, câmpul *Cod\_elev* este cheie primară, iar *Cod\_clasa* – cheie externă a tabelului *Elevi*.

De regulă, fiecare tabel al unei baze de date relationale se află într-o legătură cu cel puțin un alt tabel al același baze. Există trei **tipuri de relații între tabele**:

- a) *unu la unu*; b) *unu la mulți*; c) *mulți la mulți*.

Într-o relație de tipul ***unu la unu*** (se notează 1:1), fiecarei înregistrări a unui tabel îi poate corespunde cel mult o înregistrare a celuilalt tabel și invers. Această tip de relație se obține atunci când un tabel, având prea multe câmpuri, este divizat în două tabele. În acest caz, câmpurile primare ale tabelelor vor avea date identice.

De exemplu, între tabelele *Elevi* și *Adrese\_elevi* a fost creată o relație de tipul *unu la unu*. Observăm că ambele tabele au același câmp cheie primară *Cod\_elev* cu date identice.

Tabelul **Adrese\_Elevi**

<b>Cod_elev</b>	<b>Loc_elev</b>	<b>Strada_elev</b>	<b>Nr_casa_elev</b>	<b>Ap_elev</b>
e001	Chișinău	Mihail Sadoveanu	40	23
e002	Stăuceni	Viei	9	

Schematic, această relație este reprezentată în figura 4.2.

În cazul relației ***unu la mulți*** (se notează 1:∞ sau 1:M), fiecarei înregistrări a unui tabel îi pot corespunde câteva înregistrări ale celuilalt tabel, iar fiecarei înregistrări din tabelul al doilea îi poate corespunde cel mult o înregistrare a primului tabel. Pentru a stabili o astfel de relație, primul tabel trebuie să conțină o cheie primară, iar cel de-al doilea – o cheie externă de tip compatibil cu tipul cheii primare a primului tabel.

De exemplu, tabelul *Clase* se află în relația *unu la mulți* cu tabelul *Elevi*, deoarece fiecarei clase „îi corespund câțiva” elevi, iar fiecare elev „apartine” unei singure clase.

Tabelul **Clase**

<b>Cod_clasa</b>	<b>Anul_de_studii</b>	<b>Nume_clasa</b>	<b>Cod_profil</b>
c01	10	A	p1
c02	10	B	p1
c03	10	C	p2
c04	10	D	p2

Cheia primară *Cod\_clasa* a tabelului *Clase* este „compatibilă” cu cheia externă *Cod\_clasa* a tabelului *Elevi*. Schematic, această relație este reprezentată în figura 4.3.

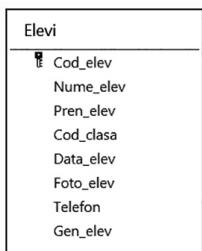


Fig. 4.2. Relația 1:1

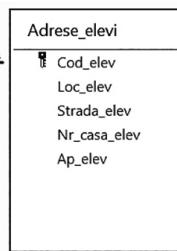


Fig. 4.3. Relația 1:M

**Observație:** Cheia externă în cazul relației unu la mulți se mai numește *cheie străină*.

Într-o relație **mulți la mulți** (se notează  $\infty:\infty$  sau M:M), fiecarei înregistrări a unui tabel îi pot corespunde câteva înregistrări ale celuilalt tabel și invers. De exemplu, tabelele *Clase* și *Discipline* se află în relația *mulți la mulți*, deoarece în fiecare clasă se predau mai multe discipline și fiecare disciplină poate fi predată în câteva clase.

Tabelul **Discipline**

Cod_disciplina	Nume_disciplina
d01	Matematica
d02	Informatica
d03	Chimia
d04	Fizica

Tabelul **Prof\_dis\_clasa**

Cod_repart	Cod_clasa	Cod_dis	Cod_prof
r001	c01	d01	prof01
r002	c01	d02	prof01
r003	c01	d03	prof02
r004	c01	d05	prof04

Relația *mulți la mulți* dintre două tabele se poate stabili prin două relații de tipul *unu la mulți*. Fiecare dintre cele două tabele se leagă într-o relație *unu la mulți* cu un al treilea tabel.

De exemplu, relația *mulți la mulți* dintre tabelele *Clase* și *Discipline* se realizează prin tabelul *Prof\_dis\_clasa* (fig.4.4).

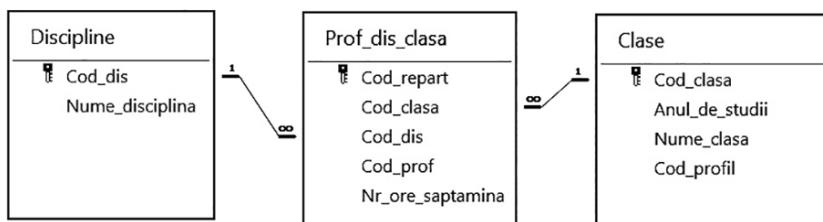


Fig. 4.4. Stabilirea relației M:M prin relațiile 1:M și M:1

**Observație:** Prima înregistrare din tabelul *Prof\_dis\_clasa* informează că în clasa *c01* (adică în clasa a X-a A) disciplina *d01* (adică *Matematica*) este predată de profesorul *prof01* (datele despre acest profesor se păstrează în tabelul *Profesori*).

## Principii de proiectare

Proiectând o bază de date relațională, programatorii definesc tabelele astfel încât ele să respecte anumite principii (de exemplu, să fie aduse la aşa-numitele *forme normale*).

- Unul dintre aceste principii presupune că datele câmpurilor tabelelor trebuie să fie **date elementare**. Din aceste considerente, adresa unui elev sau a unui profesor se păstrează în câteva câmpuri. Este evident că prelucrarea înregistrărilor ar fi fost mai dificilă dacă adresa era stocată într-un singur câmp.
- Pentru ca actualizarea înregistrărilor din tabele să fie optimală (de exemplu, să fie mai rapidă), se recomandă excluderea multiplă a repetărilor valorilor fiecărui câmp (excepție făcând cazul câmpurilor legate cu chei primare ale altui tabel) prin **descompunerea tabelului** respectiv în câteva tabele.

De exemplu, datele despre profesori și despre disciplinele pe care ei le predau în clase sunt stocate în patru tabele: *Profesori*, *Discipline*, *Clase* și *Prof\_dis\_clase*. Așa cum între aceste tabele există relații, pentru a modifica în toate înregistrările numele unui profesor (înregistrări obținute la căutarea, extragerea sau prelucrarea informației din aceste tabele), este suficient să efectuăm schimbarea numelui doar în tabelul *Profesori*.

## Întrebări și exerciții

- Sistematizează-ți cunoștințele!** Numeți etapele de proiectare a unei baze de date.
- Explică!** Care sunt entitățile unei baze de date relaționale?
- Explică!** Explicați principiile de proiectare a unei baze de date relaționale.
- Analizează!** Prin ce se deosebește relația M : M de relația 1 : M?
- Exersează!** Examinați tabelul *Elevi* al bazei de date *Liceu* și stabiliți tipul valorilor fiecărui câmp al acestui tabel.
- Aplică!** Alegeți o cheie primară pentru tabelul:
  - Clasa\_12* cu câmpurile *Număr\_de\_ordine*, *Nume\_elev*, *Prenume\_elev*, *Telefon\_elev*;
  - Angajați* cu câmpurile *Nume\_angajat*, *Prenume\_angajat*, *Vârsta\_angajat*, *Gen\_angajat*, *Număr\_buletin\_de\_identitate\_angajat*;
  - Parcare\_auto* cu câmpurile *Marca\_auto*, *Număr\_de\_înmatriculare\_auto*, *Foto\_auto*, *Nume\_proprietar\_auto*;
  - Tări* cu câmpurile *Nume\_țară*, *Suprafață\_țară*, *Capitală\_țară*, *Număr\_locuitori\_țară*.
- Analizează!** Ce tip de relație se poate stabili între tabelele:
  - Vinuri* și *Producători\_de\_vinuri*;
  - Cărți* și *Autori*;
  - Poezii* și *Autori*;
  - Specialități* și *Universități*?
- Aplică!** Proiectați tabelele bazei de date care va conține informații:
  - dintr-o agendă de telefoane;
  - despre cărțile bibliotecii personale;
  - despre automobile;
  - despre mariile personalități ale țării noastre.

## 4.4

# Sisteme de gestiune a bazelor de date

## Concepțe generale despre sisteme de gestiune a bazelor de date

Un **sistem de gestiune a bazelor de date** (SGBD) este un ansamblu de programe care permit construirea bazelor de date, introducerea și actualizarea datelor, asigurarea controlului de acces al acestor date, gestionarea informațiilor din baza de date, precum și crearea aplicațiilor cu baze de date. Un SGBD exercită următoarele funcții:

- a) definește *baza de date*, în sensul că descrie tipurile și structurile de date, relațiile dintre ele și modalitățile de accesare a informațiilor din bază;
  - b) actualizează *baza de date*, adică permite inserarea, redactarea și ștergerea datelor;
  - c) execută *interrogări către baza de date*, în urma cărora informațiile se sortează sau se filtrează după anumite criterii formulate de utilizatori;
  - d) creează *date noi*, care se obțin în baza efectuării unor calcule, inclusiv a totalizațiilor;
  - e) creează *rapoarte simple și complexe* sub formă de tabele, grafice, diagrame etc.;
  - f) asigură *întreținerea bazei de date*, care presupune repararea bazei în cazul unor defecțiuni, compactarea (sau defragmentarea) ei, precum și crearea còpiilor de siguranță atât pentru date, cât și pentru obiectele bazei;
  - g) protejează *baza de date* de accesul care nu este autorizat și stabilește această autorizare pentru acces complet sau partajat.
- Software-ul SGBD este format din:
    - a) *Dicționarul de date (Data Dictionary)*, care conține o descriere a structurii datelor folosite în baza de date;
    - b) *Limbajul de interrogare (Query Language)*, care asigură accesul la informațiile din baza de date. Cel mai răspândit limbaj de interrogare (folosit în diferite SGBD) este limbajul SQL (*Structured Query Language*).
  - Hardware-ul SGBD poate fi format:
    - a) *dintr-un singur calculator* sau
    - b) *dintr-o rețea de calculatoare*, în care pe calculatorul principal (*serverul*) se află programele-componente din SGBD care administrează și controlează accesul către baza de date, iar pe celelalte calculatoare – programele destinate utilizatorilor.

Cele mai răspândite SGBD sunt Microsoft Office Access, Oracle, MySQL, PostgreSQL, SQLite, Microsoft SQL Server etc.

## Sistemul de gestiune a bazelor de date Microsoft Office Access

Microsoft Office Access (în continuare vom spune doar Access) este un sistem de gestiune a bazelor de date relaționale care funcționează în mediul Windows. Prima sa versiune a fost lansată în anul 1993. Cu ajutorul lui pot fi create baze de date complexe, având pentru acest scop tot instrumentarul necesar. Comparativ cu alte SGBD-uri, este comod și simplu în utilizare. Pentru crearea interrogărilor, formularelor, rapoartelor și a altor produse de baze de date, Access pune la îndemâna programatorilor și utilizatorilor un set de programe de asistență, care automatizează diferite etape de lucru.

Sistemul Access poate fi lansat în câteva moduri:

- executând un dublu-clic pe referința Access de pe suprafața de lucru Windows;
- selectând opțiunea *Microsoft Office Access* cu ajutorul meniului *Start*;
- executând un dublu-clic pe pictograma oricărui fișier cu extensia .mdb.

Fiind lansat Access, putem **crea o bază de date**, selectând *File* → *New* → *Blank database...*

Apare fereastra *Blank database*. În caseta *File name* scriem numele bazei de date pe care dorim să o creăm. Butonul care însoțește caseta din dreapta permite să selectăm catalogul în care vom păstra această bază de date. Apăsăm butonul *Create*.

În interiorul ferestrei Access apare **fereastra bazei de date** cu numele ei.

Suprafețele de lucru ale variatelor versiuni ale sistemului Access nu sunt total diferite. Fiecare dintre aceste suprafețe are următoarele elemente de bază:

- bara de titlu Access;
- bara cu meniuri (*File*, *Home*, *Create*, *External Data*, *Database Tools*, *Help* etc.) a ferestrei Access;
- bara cu instrumente Access;
- fereastra bazei de date cu opțiuni (în versiunile moderne se numește *All Access Objects*) pentru crearea și gestionarea diferitelor clase de obiecte Access (*Tables* – tabele, *Queries* – interogări, *Forms* – formulare, *Reports* – rapoarte etc.).

În continuare vom explica cum se elaborează și se gestionează cu Access o bază de date. Vom crea și vom prelucra baza de date *Liceu*, care va păstra informații despre elevii, elevele, profesoarele și profesorii unui liceu.

## Structura bazei de date *Liceu*

Baza de date *Liceu* va fi formată din 8 tabele legate între ele (fig. 4.5).

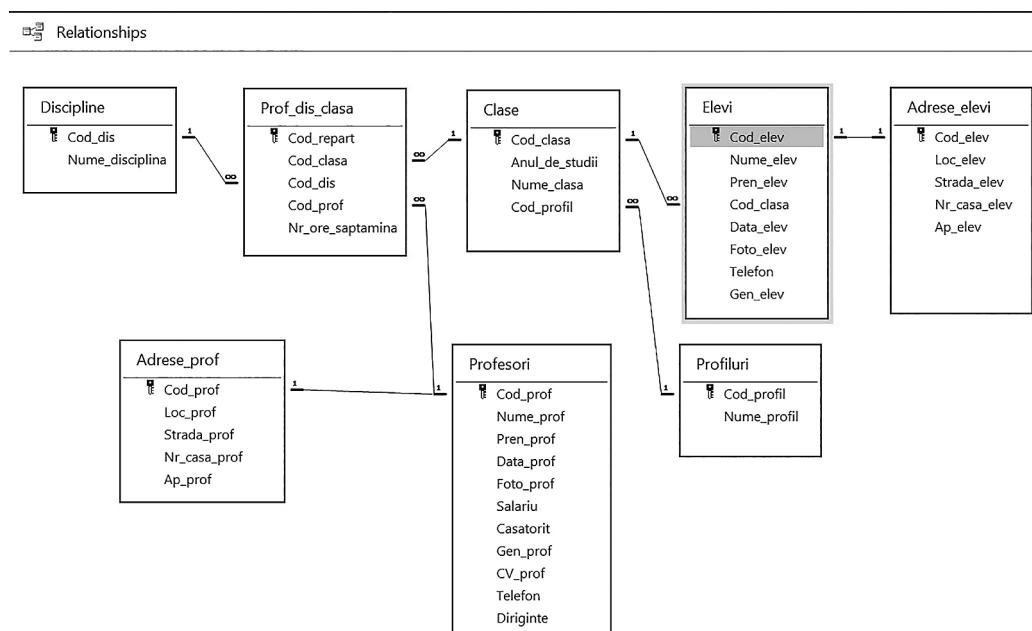


Fig. 4.5. Tabelele bazei de date *Liceu*

Prezentăm structura celor 8 tabele. Atragem atenția că aici sunt arătate doar câteva înregistrări ale fiecărui dintre *tabelele 1–8*.

Tabelul 1. Profiluri

Cod_profil	Nume_profil
p1	Real
p2	Umanist

Tabelul 2. Discipline

Cod_dis	Nume_disciplina
d01	Matematica
d02	Informatica

Tabelul 3. Profesori

Cod_prof	Nume_prof	Pren_prof	Data_prof	Foto_prof	Salariu	CV_prof	Casatorit	Gen_prof	Telefon	Diriginte
prof01	Albu	Ion	01.03.1968	Package	12880	Grad...	Yes	M	64-41-72	c02
prof03	Dodon	Ana	09.07.1958	Package	8040	S-a năs...	No	F	51-65-70	c01

Tabelul 4. Adrese\_prof

Cod_prof	Loc_prof	Strada_prof	Nr_casa_prof	Ap_prof
prof01	Cricova	Vinului	89	
prof02	Chișinău	Grenoble	81	77

Tabelul 5. Clase

Cod_clasa	Anul_de_studii	Nume_clasa	Cod_profil
c01	10	A	p1
c02	10	B	p1

Tabelul 6. Elevi

Cod_	Elev	Pren_elev	Cod_clasa	Data_elev	Foto_elev	Telefon	Gen_elev
e001	Bacinschi	Sabina	c01	28.09.2007	Package	29-82-54	F
e002	Belobrov	Andreea	c01	18.10.2007	Package	44-26-48	F

Tabelul 7. Adrese\_elevi

Cod_elev	Loc_elev	Strada_elev	Nr_casa_elev	Ap_elev
e001	Chișinău	Mihail Sadoveanu	40	23
e002	Stăuceni	Viei	9	

Tabelul 8. Prof\_dis\_clasa

Cod_repart	Cod_clasa	Cod_dis	Cod_prof
r001	c01	d01	prof01
r002	c01	d02	prof01

**Observație:** Tabelele bazei de date *Liceu* pot fi descărcate de pe adresa [www.ctice.gov.md](http://www.ctice.gov.md)

## Întrebări și exerciții

- 1 **Sistematizează-ți cunoștințele!** Descrieți structura unui sistem de gestiune a bazelor de date.
- 2 **Explică!** Explicați funcțiile sistemelor de gestiune a bazelor de date.
- 3 **Explică!** Cum poate fi lansat SGBD Access?
- 4 **Aplică!** Examinați suprafața de lucru a SGBD Access și caracterizați clasele de obiecte Access.
- 5 **Aplică!** Observați înregistrările din tabelele bazei de date *Liceu* și stabiliți:
  - a) profilul clasei în care învață eleva Sabina Bacinschi;
  - b) numele profesorului care predă Informatica în clasa a X-a A;
  - c) denumirea unei discipline predate de profesorul Ion Albu;
  - d) adresa dirijintelui clasei a X-a A.

## 4.5 Crearea tabelelor

Am menționat deja că entitățile unei baze de date Access sunt tabelele. Cu ajutorul lor se pot crea celelalte obiecte ale bazei de date: interrogări, formulare, rapoarte etc. Prin urmare, o bază de date relațională nu are sens fără existența tabelelor.

### Proiectarea structurii unui tabel

Prin **structura unui tabel** se subînțelege informația care descrie câmpurile tabelului: denumirea lor, tipul și proprietățile fiecărui câmp, câmpurile care constituie cheia primară etc.

Numele implicit al noului tabel este *Table 1*. El poate fi redenumit prin comanda *Rename* din meniul contextual al pictogramei tabelului.

Instrumentul *View* de sub meniul *File* permite crearea, editarea sau vizualizarea tabelului în două moduri: *Design view* (regim de proiectare) și *Datasheet View* (regim implicit, de foaie de calcul).

1. Alegem regimul *Design view*.

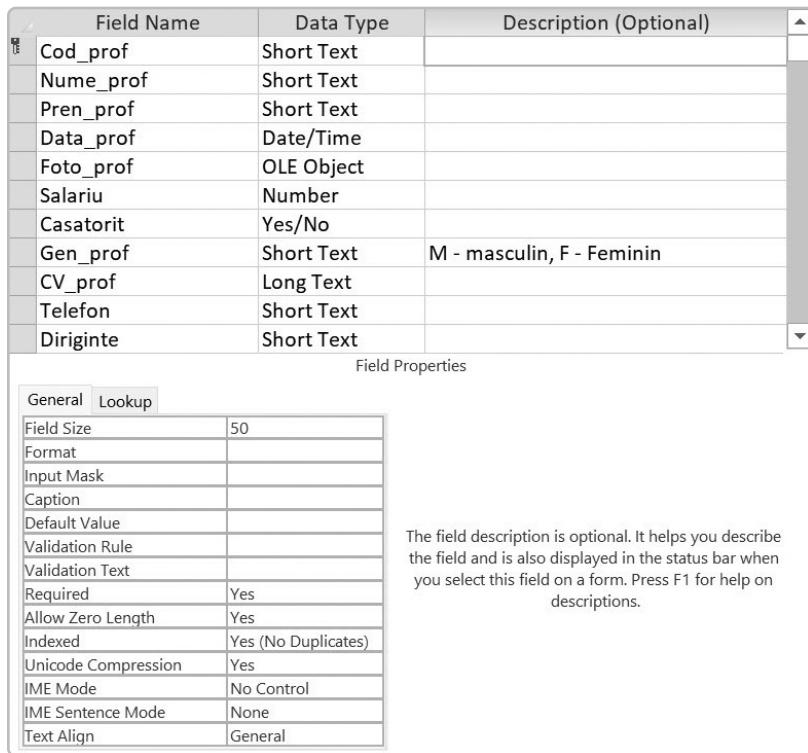


Fig. 4.6. Fereastra de dialog Tabelul Profesori

2. Apare fereastra de dialog de tip *Table* (fig. 4.6). Ea este formată din două zone:

- a) zona pentru descrierea structurii documentului;
- b) zona descrierii proprietăților câmpului selectat din prima zonă (*Field Properties*).

Zona pentru descrierea structurii documentului este divizată în trei coloane:

- *Field Name* (identificatorul câmpului);
- *Data Type* (tipul câmpului, adică al valorilor lui);
- *Description* (descrierea câmpului).

Astfel, pentru fiecare câmp al tabelului ce urmează a fi creat se precizează identificatorul, tipul și descrierea câmpului.

**Identificatorul câmpului** poate conține diferite caractere, în afară de semnele „.”, „!”, „[”, „]”, spațiul de debut și caracterele care nu sunt vizibile (de exemplu, returnul de car).

Lungimea identificatorului câmpului nu poate fi mai mare decât 64 de caractere.

**Tipurile câmpurilor** acceptate de Access sunt prezentate în următorul tabel:

Denumirea tipului	Descrierea valorilor tipului
Short Text	Șiruri de caractere alfanumerice. Până la 255 de caractere
Long Text (sau Memo)	Texte mari. Până la 63 999 de caractere
Number	Numere
Large Number	Numere mari

Denumirea tipului	Descrierea valorilor tipului
Date/time	Date calendaristice
Currency	Valori monetare
AutoNumber	Numerele naturale 1, 2, 3, ... inserate în ordine și în mod automat la adăugarea unei înregistrări
Yes/No	Valorile Yes sau No
OLE Object	Imagini sau sunete
Hiperlink	Adrese Web
Attachement	Fișier
Calculated	Câmp calculat
Lookup Wizard	Valori dintr-un tabel sau dintr-o listă de valori

**Descrierea câmpului** (nu este obligatorie) poate conține note explicative referitor la câmp. Proprietățile câmpurilor vor fi descrise mai târziu.

3. După ce am definit câmpurile tabelului, **stabilim o cheie primară** pentru tabel. Cu acest scop selectăm câmpul necesar (ale căruia valori nu se vor repeta), apoi alegem comanda *Primary Key* din meniul flotant al câmpului (sau executăm un clic pe instrumentul *Primary Key*  de pe bara cu instrumente Access). În tabelul din figura 4.6, câmpul *Cod\_prof* a fost ales drept cheie primară. Atunci când nu definim o cheie primară, sistemul Access va sugera stabilirea unei astfel de chei imediat după salvarea tabelului.

Dacă utilizatorul va accepta sugestia, atunci sistemul va stabili drept cheie primară primul câmp de tip *AutoNumber* sau (dacă tabelul nu conține un astfel de câmp) va crea un astfel de câmp (cu numele implicit *ID*).

4. **Salvăm tabelul** selectând comanda *Save* din meniul *File*.

**Exercițiu:** Observați denumirile și tipurile câmpurilor tabelului din figura 4.6. Creați similar tabelul *Elevi* al bazei de date *Liceu*, descrisă în capitolul precedent.

## Proprietățile câmpurilor tabelului

Proprietățile câmpului sunt caracteristici care stabilesc un control suplimentar asupra modului în care sunt memorate, introduse sau afișate datele lui. Aceste proprietăți depind de tipul câmpului și se precizează în zona inferioară a ferestrei tabelului (fig. 4.6).

1. Proprietatea **Field Size** determină formatul mărimeii datelor câmpului și există doar pentru câmpurile cu numere sau texte.

- Pentru tipul *Short Text* se acceptă valori de la 0 la 255, stabilind astfel limita de lungime a sirului de caractere ce va fi memorat în câmp. Valoarea 50 este implicită.
- Pentru tipul *Number* se poate alege una dintre valorile *Byte*, *Integer*, *Long integer* (valoare implicită), *Single*, *Double*, *Replication ID*, *Decimal*.

2. Proprietatea **Format** particularizează printr-un şablon modul în care vor fi afişate datele câmpului și există pentru toate tipurile, cu excepția tipului OLE Object.
- În cazul tipurilor *Text* și *Memo*, şablonul poate fi creat cu ajutorul următoarelor simboluri:

Simbol	Descriere
@	Caracter de text sau spațiu.
&	Caracterul de text nu este obligatoriu.
<	Toate caracterele vor fi minuscule.
>	Toate caracterele vor fi majuscule.
-	Afișează caracterul -.

**Exemplu:** Şablonul @@-@@-@@@> va afișa în loc de textul *abcdef* textul *AB-CD-EF*.

- Pentru tipurile *Number* și *Currency* se poate selecta una dintre valorile *General number*, *Currency*, *Euro*, *Fixed*, *Standard*, *Percent* sau *Scientific*.
- Formatul tipului *Date/Time* poate fi *General Date*, *Long Date*, *Medium Date*, *Short Date*, *Long Time*, *Medium Time*, *Short Time*. Precizăm că dacă anul este scris cu două cifre, atunci pentru valori din intervalul [00, 29] Access subînțelege anii 2000–2029, iar pentru valori din intervalul [30, 99] – anii 1930–1999.

**Exemplu:** Formatul *Long Date* va afișa data 28.11.89 astfel: 28 noiembrie 1989.

- Tipul *Yes/No* acceptă una dintre formatele *Yes/No*, *On/Off*, *True/False*. În ultimul caz, utilizatorii vor putea scrie în câmp și valorile 1, respectiv, 0.
3. Proprietatea **Input Mask** se folosește la elaborarea unui şablon de restricționare a caracterelor (se mai spune *mască de intrare*) ce urmează a fi introduse în câmp. Pentru crearea şablonului sunt admise următoarele simboluri:

Simbol	Descriere
0	Cifră (nu poate fi precedată de + ori -). Introducere obligatorie.
9	Cifră (poate fi precedată de + ori -) sau spațiu. Introducere optională.
#	Cifră (poate fi precedată de + ori -) sau spațiu (la salvare este eliminat). Introducere optională.
L	Literă. Introducere obligatorie.
?	Literă. Introducere optională.

Simbol	Descriere
A	Literă sau cifră. Introducere obligatorie.
a	Literă sau cifră. Introducere optională.
&	Orice caracter sau spațiu. Introducere obligatorie.
C	Orice caracter sau spațiu. Introducere optională.
.,:;- /	Separatori pentru date calendaristice sau pentru clasele numărului (unități, mii, milioane, miliarde etc.). Separatorul predefinit depinde de setările din fereastra <i>Regional Setings</i> (poate fi afișată din Panoul de control Windows).
<	Transformă simbolurile-litere din dreapta în minuscule.
>	Transformă simbolurile-litere din dreapta în majuscule.
!	Forțează introducerea datelor de la dreapta spre stânga.
\	Afișează doar caracterul care urmează după \ (de exemplu, şablonul \M afișează litera M).
"Şir de caractere"	Afișează doar şirul de caractere (fără ghilimele).
Password	În loc de simbolurile introduse, se vor afișa caractere.

**Exemplu:**

- a) Şablonul >L<L0\\$ acceptă şiruri de caractere din 4 simboluri, dintre care primul este o literă majusculă, al doilea – o literă minusculă, al treilea – o cifră, ultimul caracter fiind litera S.
- b) Pentru scrierea numerelor de telefon doar de forma (+373 22) XX-XX-XX se va folosi şablonul „(+373 22) 00\00\00”.

Pentru crearea şablonelor se poate folosi programul *Input Mask Wizard* (se lansează prin executarea unui clic pe butonul  din caseta proprietății *Input Mask*), care oferă 10 formate uzuale ale măștilor de intrare.

4. Proprietatea **Caption** specifică textul care va fi afișat în calitate de denumire a câmpului în interogări, formulare sau rapoarte. Dacă această proprietate nu se completează, atunci Access va folosi în calitate de nume identificatorul câmpului.
5. Proprietatea **Default Value** stabilește valoarea implicită a câmpului. Apare automat la inserarea unei noi înregistrări.
6. Cu ajutorul proprietății **Validation Rule** se pot forma condiții de validare a datelor care urmează a fi introduse în câmp. Condițiile de validare reprezintă expresii care se scriu utilizând operatori și funcții Access sau VBA (*Visual Basic for Applications*). De exemplu, condiția  $>= 100$  permite utilizatorului să introducă într-un câmp numeric doar numere mai mari sau egale cu 100.
7. În proprietatea **Validation Text** se scrie textul care va apărea într-o fereastră de avertizare, dacă valoarea introdusă în câmp nu respectă condițiile de validare din *Validation Rule*.
8. Proprietatea **Required** acceptă doar valorile Yes și No. Valoarea Yes obligă utilizatorul să completeze câmpul. Este inutil de completat această proprietate pentru cheile primare

(deoarece un astfel de câmp nu acceptă valori vide) sau dacă condiția de validare este *Is Not Null* (nu este nulă).

9. În proprietatea **Allow Zero Length**, de asemenea, se poate scrie doar una dintre valorile *Yes* sau *No*. Ea există doar pentru câmpurile de tip *Text* și *Memo*. Pentru *Yes* câmpul va accepta valori de lungime 0, adică șiruri vide de caractere, chiar dacă proprietatea *Required* va fi *Yes*.
10. Proprietatea **Indexed** permite (pentru indexul *Yes (Duplicates Ok)*) sau interzice (pentru indexul *Yes (No Duplicates)*) repetarea valorilor în câmp. Indexul existent poate fi eliminat dacă din lista derulantă a proprietății se alege valoarea *No*. Pentru o cheie primară, indexul *Yes (No Duplicates)* va apărea automat (fig. 4.6) și nu va putea fi modificat.
11. Proprietatea **New Values** se aplică doar câmpurilor de tip *AutoNumber*. Pentru valoarea *Increment* sistemul Access va genera valori noi în câmp, adăugând 1 la cea mai mare valoare existentă. Dacă atribuim proprietății *New Values* valoarea *Random*, atunci câmpul va fi completat cu valori generate aleator.

## Întrebări și exerciții

- 1 **Aplică!** Examinați tabelul *Profesori* al bazei de date *Liceu* și descrieți structura lui.
- 2 **Explică!** Care sunt tipurile de date ce pot fi păstrate într-o bază de date Access?
- 3 **Explică!** Care trebuie să fie tipul unui câmp pentru a putea stoca în el fotografii? Dar adrese de pagini Web? Biografia unei personalități?
- 4 **Aplică!** Ce trebuie să facem pentru ca fiecare înregistrare nouă să nu fie acceptată în tabel, dacă utilizatorul nu a completat ultimele două câmpuri ale lui?
- 5 **Analizează!** Un câmp care nu este cheie primară nu acceptă repetări de valori în diferite înregistrări. Care este cauza?
- 6 **Explică!** Cum putem verifica dacă în baza de date *Liceu* există informații despre elevi cu aceeași zi de naștere?
- 7 **Aplică!** Cărui an îi aparține data:  
a) 01.01.01;    b) 30.12.30;    c) 17.12.89;    d) 15.04.28?
- 8 **Elaborează!** Creați un tabel Access care va conține informații despre o colecție de muzică. Includeți cel puțin câmpurile pentru memorarea numelui interpretului, a firmei producătoare de înregistrări, a anului apariției, a formatului (disc, casetă, CD etc.) și a valorii de evaluare atribuite piesei muzicale, potrivit preferințelor (de exemplu, de la 5 la 10).
- 9 **Elaborează!** Creați un tabel Access care va conține date despre o colecție de rețete culinare. Includeți cel puțin câmpurile în care se vor stoca denumirile bucatelor, tipurile lor (felurile întâi, garnituri, fripturi, copturi, deserturi etc.), timpul de pregătire, originea bucatelor (moldoveniști, franțuzești, japoneze etc.).
- 10 **Aplică!** Alcătuiți un şablon care va obliga introducerea într-un câmp numeric doar a numerelor întregi:  
a) din intervalul [10... 99];    b) de trei cifre;    c) negative de două cifre.

11

**Aplică!** Creați un şablon care va permite introducerea într-un câmp a numerelor de identificare a buletinelor moldoveneşti. Orice buletin moldovenesc are un număr de identificare format dintr-o literă majusculă, urmată de 8 cifre.

## 4.6 Stabilirea relaţiilor dintre tabele

Cunoaştem deja că între două tabele ale unei baze de date relationale poate exista una din tre următoarele tipuri de relaţie: 1 : 1, 1 : M, M : M.

Sistemul Access foloseşte pentru afişarea şi crearea relaţiilor dintre tabele fereastra *Relationships*. Vom examina crearea relaţiilor dintre două tabele printr-un exemplu.

Să stabilim relaţia 1 : M dintre tabelele *Clase* şi *Elevi* ale bazei de date *Liceu*.

1. Executăm un clic dublu pe instrumentul *Relationships* din meniul *Database Tools*. Apare meniul *Relationships Design* şi fereastra *Relationships* în care sunt prezentate tabelele pentru care au fost stabilite relaţii.
2. Executăm un clic pe butonul *Add Tables* din meniul *Relationships Design*. Apare fereastra *Show Table* (fig. 4.7), din care selectăm pe rând tabelele *Elevi* şi *Clase*, confirmând de fiecare dată alegera prin apăsarea butonului *Add*. În fereastra *Relationships* apar identificatorii câmpurilor tabelelor selectate (fig. 4.8).

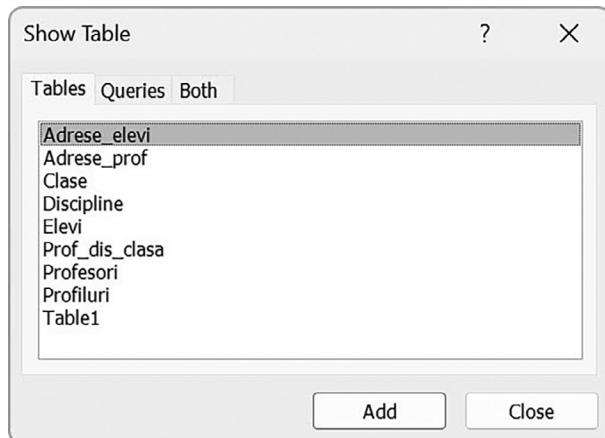


Fig. 4.7. Fereastra de dialog *Show Table* (Afisează tabelul)

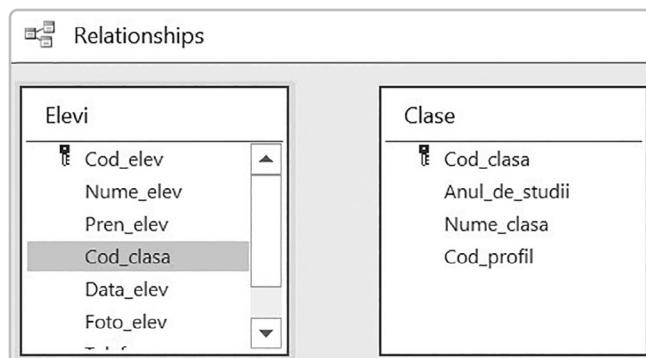


Fig. 4.8. Fereastra de dialog *Relationships* (Relaţii)

3. Selectăm câmpul *Cod\_clasa* al tabelului *Elevi*, apoi, ținând apăsat butonul mouse-ului, tragem cursorul spre câmpul *Cod\_clasa* al tabelului *Clase*. Apare fereastra *Edit Relationships*, în care automat a fost determinat tipul relației 1: M (*Relationship Type: One-To-Many*). De asemenea, putem activa următoarele caracteristici ale relației (fig. 4.9):
- Asigurarea integrității referențiale* (*Enforce Referential Integrity*);
  - Actualizarea în cascadă a înregistrărilor* (*Cascade Update Related Fields*);
  - Excluderea în cascadă a înregistrărilor* (*Cascade Delete Related Records*).

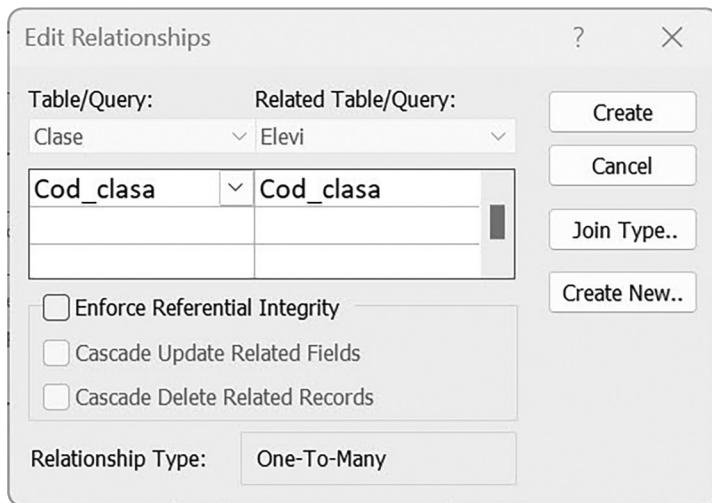


Fig. 4.9. Fereastra de dialog *Edit Relationships*  
(Editarea relațiilor)

Dacă este activată caracteristica **Asigurarea integrității referențiale**, atunci câmpul cheie externă al tabelului subordonat va accepta doar valori ale câmpului cheie primară al tabelului principal. Astfel, în câmpul *Cod\_clasa* al tabelului *Elevi* vom putea scrie doar „codurile” claselor înregistrate în tabelul *Clase*.

**Actualizarea în cascadă a înregistrărilor** înseamnă că la modificarea unei valori *V* din câmpul cheie primară al tabelului principal, automat se vor modifica corespunzător toate valorile *V* din câmpul cheie străină al tabelului subordonat. De exemplu, dacă vom schimba în tabelul *Clase* „codul” clasei a 10-a A din c01 în c001, atunci fiecare valoare c01 din câmpul *Cod\_clasa* al tabelului *Elevi* va fi substituită cu valoarea c001.

Dacă este activată caracteristica **Excluderea în cascadă a înregistrărilor**, atunci la eliminarea unei înregistrări *X* din tabelul principal se vor elimina toate înregistrările din tabelul subordonat, care conțin în câmpul cheie străină valoarea din câmpul cheie primară a înregistrării *X*. De exemplu, dacă vom exclude din tabelul *Clase* ultima înregistrare (are „codul” c12 și corespunde clasei a 12-a D), atunci din tabelul *Elevi* vor fi „eliminați” toți elevii și elevele clasei a XII-a D.

## Întrebări și exerciții

- 1 Explicați algoritmul de stabilire a relației dintre două tabele Access.
- 2 **Sistematizează-ți cunoștințele!** Ce înseamnă *integritatea referențială a datelor*?
- 3 **Aplică!** Disciplinele școlare aparțin următoarelor arii curriculare:

- a) limbă și comunicare;
- b) educație socioumanistică (istorie, geografie, educație civică);
- c) matematică și științe (matematică, fizică, chimie, biologie);
- d) tehnologii (informatică);
- e) sport (educație fizică).

Deschideți baza de date *Liceu*. Creați și completați în ea tabelul *Arii curriculare*, apoi stabiliți o relație dintre acest tabel și tabelul *Discipline*.

- 4 **Explică!** Care este rolul caracteristicii *Cascade Update Related Fields* a unei relații?
- 5 **Explică!** Cu ce scop se utilizează caracteristica *Cascade Delete Related Records* a unei relații?

## 4.7 Modificarea tabelelor

### Introducerea și editarea datelor

Pentru a introduce sau a edita date într-un tabel, deschidem acest tabel în regimul *Datasheet View* (poate fi selectat cu instrumentul *View* de sub meniul *File*).

Comutarea dintre regimurile *Datasheet View* (regim de editare, fig. 4.10) și *Design View* (regim de proiectare) se realizează prin instrumentul *View* de sub meniul *File*.

La crearea sau modificarea conținutului unei înregistrări, la stânga ei apare un **selector de înregistrare (SI)**, al cărui aspect depinde de starea înregistrării (vezi *tabelul* care urmează).

Elevi							
	Cod_elev	Nume_ele	Pren_elev	Cod_clasă	Data_elev	Foto_elev	Telefon
<input type="checkbox"/>	e217	Pintilei	Pavel	c08	03.05.2006		022-47-69-44
<input type="checkbox"/>	e218	Pogujanschii	Alexandru	c08	03.11.2006		022-59-58-19

Fig. 4.10. Tabelul Elevi afișat în regim de editare

SI	Starea înregistrării
<input type="checkbox"/>	Înregistrarea curentă este selectată.
<input checked="" type="checkbox"/>	Înregistrare nouă, în care se pot introduce date.
<input type="checkbox"/>	Utilizatorul editează înregistrarea, iar modificările nu sunt încă salvate.
<input type="checkbox"/>	Înregistrarea este blocată de alt utilizator și nu se poate edita (cazul mediului multi-utilizator – mediu în care mai multe persoane pot utiliza simultan baza de date).

Pentru gestionarea rapidă a înregistrărilor se pot utiliza instrumentele din partea de jos a ferestrei tabelului:  285 of 285, având, respectiv, următoarele funcții (de la stânga spre dreapta):

- activarea primei înregistrări;
- activarea înregistrării predecesoare celei curente;
- afișarea numărului de ordine al înregistrării curente sau activarea înregistrării cu numărul de ordine din casetă;
- activarea înregistrării succesoare celei curente;
- activarea ultimei înregistrări;
- adăugarea unei înregistrări noi.

Introducerea și editarea datelor unui tabel se fac prin metode caracteristice lucrului cu texte. De exemplu, se pot copia date utilizând memoria *Clipboard*, iar ștergerea lor se poate realiza folosind tastele *Backspace* și *Delete*.

O singură apăsare pe tasta *Esc* anulează acțiunile de modificare a informației din câmpul curent, iar o dublă apăsare pe ea – acțiunile de modificare a înregistrării curente.

Nu este obligatoriu de completat toate câmpurile unei înregistrări, cu excepția câmpului cheie primară (care nu poate conține valori vide) și a celor care au setată cu Yes proprietatea *Required*.

Un câmp poate fi selectat prin executarea unui clic pe *antetul* lui (celula care afișează identificatorul câmpului), iar o înregistrare – prin executarea unui clic pe *antetul* ei (celula în care apare selectorul de înregistrare). Dacă după selectarea unui câmp (sau a unei înregistrări) nu eliberăm butonul mouse-ului și-l poziționăm pe următorul câmp (respectiv pe următoarea înregistrare), vor deveni selectate ambele câmpuri (respectiv ambele înregistrări).

Menționăm că **asupra datelor sau înregistrărilor se pot efectua următoarele operații:**

- adăugarea unei înregistrări noi înaintea celei curente (comanda *New Record* din meniul flotant al antetului înregistrării sau din meniul *Insert* al meniului principal Access);
- ștergerea înregistrării curente (comanda *Delete Record* din meniul flotant al antetului înregistrării sau din meniul *Edit* al meniului principal Access);
- completarea câmpurilor;
- copierea conținutului unei celule (comenzile *Copy* și *Paste* din meniul flotant al celulei);
- copierea unei înregistrări (comenzile *Copy* și *Paste* din meniul flotant al antetului înregistrării sau din meniul *Edit* al meniului principal Access).

#### **Observații:**

1. În urma ultimei operații, utilizatorul va fi obligat să modifice valoarea celulei din câmpul cheie primară.
2. Unele dintre operațiile mentionate se pot efectua și cu ajutorul instrumentelor de pe bara de instrumente Access.
3. Pentru introducerea rapidă a datelor, se pot folosi comenzile rapide lansate cu ajutorul combinațiilor de taste.
4. Modificările unei înregistrări sunt automat salvate de Access atunci când se trece la o altă înregistrare sau la închiderea tabelului.
5. La completarea tabelelor cu înregistrări, mai întâi se vor introduce date în tabelele principale, apoi în cele subordonate.

## Modificarea aspectului tabelului

Sistemul Access afișează în mod implicit datele tabelului respectând anumite caracteristici. De exemplu, înregistrările tabelului apar ordonate crescător automat după valorile câmpului cheie primară, iar câmpurile lui se succed în ordinea în care au fost create.

Utilizatorul poate schimba modul de prezentare a informației dintr-un tabel. Mai exact, sunt permise următoarele **acțiuni de modificare a aspectului tabelului**:

- a) schimbarea ordinii de afișare a câmpurilor;
- b) modificarea ordinii de afișare a înregistrărilor;
- c) schimbarea înălțimii înregistrărilor sau lățimii câmpurilor;
- d) ascunderea coloanelor;
- e) filtrarea înregistrărilor.

Schimbarea ordinii de afișare a câmpurilor se face prin deplasarea lor. Pentru a deplasa unul sau câteva câmpuri consecutive, selectăm aceste câmpuri, apoi, ținând apăsat butonul stâng al mouse-ului, poziționăm indicatorul lui pe câmpul în fața căruia dorim să mutăm câmpurile selecțiate.

Pentru ca înregistrările unui tabel să apară ordonate crescător (respectiv descreșcător) după valorile unui câmp, selectăm acest câmp, apoi executăm un clic pe butonul din meniu (fila) *Home* (respectiv pe butonul ). Similar se face ordonarea după valorile cătorva câmpuri consecutive. Menționăm că, în acest caz, ordonarea se va face de la stânga spre dreapta, adică valorile din câmpul din dreapta se vor lua în considerare doar în cazul în care vor coincide valorile câmpului din stânga.

Acțiunile de modificare a înălțimii înregistrărilor sau a lățimii câmpurilor sunt similare cu acțiunile de schimbare a înălțimii rândurilor sau a lățimii coloanelor unui tabel într-un redactor de texte ori într-un procesor tabelar.

Pentru a ascunde o coloană, o selectăm, apoi alegem comanda *Hide Fields* din meniul ei flotant. Pentru a reafîsa coloanele ascunse, selectăm comanda *Unhide Fields* din același meniu. Apare fereastra *Unhide Columns* în care alegem coloanele necesare.

Filtrarea înregistrărilor, adică selectarea acelor înregistrări care respectă anumite condiții, se poate face prin crearea unui **filtru**. Pentru aceasta se selectează coloana (cu un clic pe eticheta ei), apoi se execută un clic pe butonul *Filter* din meniul *Home*. În fereastra apărută se stabilesc criteriile de selecție (aidoma filtrării datelor în Excel).

Un filtru poate fi înălțat selectând opțiunea *Clear filter...* din fereastra de filtrare. Pentru aceasta se selectează coloana (cu un clic pe eticheta ei), apoi se execută un clic pe butonul *Filter* din meniul *Home*. În fereastra apărută se stabilesc criteriile de selecție (aidoma filtrării datelor în Excel).

### **Observații:**

1. Modificările aspectului tabelului nu sunt automat salvate de Access la închiderea tabelului. Utilizatorul le poate salva executând un clic pe butonul *Save* de pe bara de instrumente sau lansând comanda *Save* din meniul *Edit*.
2. Modificările aspectului tabelului nu afectează structura tabelului.

## Modificarea structurii tabelului

Este firesc ca pe parcursul proiectării (sau chiar al gestionării) unei baze de date să apară necesitatea schimbării structurii unor tabele.

**Atenție!** Aceste schimbări pot afecta integritatea informațiilor din baza de date. De exemplu, micșorarea dimensiunii unui câmp de tip text poate atrage după sine trunchierea valorilor acestui câmp, iar eliminarea unui câmp cheie primară poate duce la ștergerea înregistrărilor tabelului subordonat. De fapt, în funcție de caracteristicile relațiilor dintre tabele, sistemul Access poate să nu accepte unele schimbări ale structurii tabelului.

Astfel, procesul de modificare a câmpurilor cheie primară sau a câmpurilor cheie străină se va face doar după ce vor fi distruse relațiile dintre tabele, urmând a fi restabilite ulterior. Pentru a modifica structura unui tabel, acesta trebuie deschis în regimul *Design View* (poate fi selectat cu instrumentul *View* de sub meniul *File*).

În acest regim sunt posibile următoarele acțiuni:

- a) adăugarea unui câmp nou;
- b) eliminarea unui câmp;
- c) schimbarea identificatorului câmpului;
- d) modificarea proprietăților câmpurilor;
- e) stabilirea unei chei primare;
- f) eliminarea unei chei primare.

Acțiunile de modificare a structurii unui tabel sunt similare cu acțiunile de creare a tabelului.

## Caracteristica *Lookup* a câmpurilor

Caracteristica *Lookup* permite înlocuirea casetelor de text ale câmpurilor cu liste derulante. Astfel, utilizatorul va putea completa un câmp selectând o valoare dintr-o listă de valori acceptabile. Conținutul listei poate fi încărcat dintr-un câmp cheie primară al unui tabel asociat celui curent sau poate fi creat la stabilirea caracteristicii *Lookup*.

Să alcătuim o listă derulantă cu ajutorul unui program de asistență pentru câmpul *Cod\_clasa* al tabelului *Elevi*.

1. Deschidem tabelul *Elevi* în regim de proiectare. Alegem opțiunea *Lookup Wizard* din lista derulantă a câmpului *Cod\_clasa*.
2. Apare prima casetă de dialog *Lookup Wizard*. Selectăm prima opțiune, stabilind astfel că valorile listei care urmează să fie create vor fi luate dintr-un tabel asociat. Opțiunea a două precizează că valorile listei vor fi create la ceilalți pași.
3. În următoarele două ferestre alegem tabelul *Clase*, apoi câmpurile *Cod\_clasa*, *An\_de\_studii* și *Nume\_clasa*. Valorile câmpurilor alese vor apărea în listă.
4. În fereastra a patra ajustăm lățimea coloanelor listei și atribuim un nume acestei liste.

## Întrebări și exerciții

- 1 **Sistematizează-ți cunoștințele!** Ce operații pot fi efectuate asupra datelor unui tabel?
- 2 **Analizează!** Care modificări ale tabelului pot afecta structura lui?
- 3 **Aplică!** Deschideți baza de date *Liceu*. Adăugați tabelului *Adrese\_elevi* câmpul *Adresa\_Web*. Completați acest câmp pentru primele 10 înregistrări.
- 4 **Aplică!** Deschideți tabelul *Elevi* în regim de editare:
  - a) adăugați 5 înregistrări în tabel;
  - b) ștergeți penultima înregistrare;
  - c) copiați datele primei înregistrări în câmpurile ultimei înregistrări.

**5 Aplică!** Deschideți tabelul Profesori în regim de editare:

- afișați datele tabelului în ordinea: numele, prenumele, sexul, telefonul, data nașterii a profesorilor;
- ascundeți câmpurile Salariu, CV\_prof și Foto\_prof;
- reafișați câmpurile ascunse.

**6 Exerseză!** Afișați în ordine alfabetică lista elevilor bazei de date Liceu.

**7 Exerseză!** Afișați numele și prenumele elevilor în ordinea descrescătoare a vârstei lor.

**8 Exerseză!** Afișați lista profesorilor bazei de date Liceu în ordine crescătoare a salariilor lor.

## 4.8 Expresii Access

Examinând proprietatea *Validation Rule* a câmpurilor, am menționat că regulile de validare a datelor care urmează a fi introduse în câmp se scriu folosind expresii Access. După cum vom vedea ulterior, expresiile se utilizează de asemenea la formularea cererilor de căutare a datelor și la elaborarea rapoartelor.

**O expresie Access este o declarație de intenție, care conține cel puțin un operator și un operand: constantă, identificator sau funcție.**

Expresia returnează o valoare. Un **identificator** Access este numele unui obiect al bazei de date. Tabelul, câmpul, interogarea, formularul, raportul și însăși baza de date sunt obiecte. De regulă, în cadrul expresiilor identificatorii se scriu între simbolurile [ și ].

Identificatorul unui „subobiect” este format din numele clasei de obiecte și numele „subobiectului” delimitate printr-un punct sau printr-un semn de exclamare. În acest mod, fiecărui obiect îi corespunde un singur identificator în interiorul bazei de date.

Totuși, uneori, dacă nu sunt confuzii, în calitate de identificator al „subobiectului” se poate folosi doar numele lui.

**Exemplu:** [Elevi].[Nume\_elev], [Profesori].[Salariu], [Cod\_elev].

### Operatori Access

Vom examina 6 categorii de operatori Access.

- Operatorii aritmetici** (+, -, \*, /, \, Mod, ^) se aplică asupra valorilor numerice.

**Exemplu:** 15 \ 6 returnează 2, iar 15 Mod 6 returnează 3, deoarece  $15 = 6 \cdot 2 + 3$ .

- Operatorii de comparare** compară valorile a doi operanzi și returnează una dintre valorile logice *True* sau *False*. În Access se utilizează aceiași operatori de comparare și cu aceeași semnificație ca și a operatorilor relaționali din limbajul de programare Pascal: <, <=, =, >=, >, <>.

- Operatorii logici** Access se aplică asupra operanzilor logici și de asemenea coincid cu cei din Pascal: **And**, **Or**, **Not**, **Xor**.

- Operatorul de atribuire** = atribuie valoare unui obiect, unei variabile sau unei constante.

- Operatorul de concatenare** + unește două siruri de caractere în unul.

- Alți operatori.** Următorii operatori nu fac parte din categoriile precedente. Expresiile care îi conțin returnează una dintre valorile *True* (sau 1), *False* (sau 0).

Operator	Descriere
Is	Se aplică asupra valorii Null (valoarea vidă) și verifică dacă o valoare este sau nu este vidă.
Like	Stabilește dacă un sir de caractere respectă şablonul specificat de Like. Şablonul se scrie între simbolurile "și". Şablonul poate să conțină caractere de înlocuire (? pentru un caracter, # pentru o cifră, * pentru orice număr de caractere, inclusiv lipsa lor) și liste de valori. Lista de valori se scrie între simbolurile [ și ]. Dacă valorile listei sunt precedate de simbolul !, atunci se consideră toate valorile cu excepția celor precedate de !.
In	Stabilește dacă o valoare aparține unei liste de valori. Valorile listei se delimită prin simbolul ;.
Between	Stabilește dacă o valoare numerică aparține unui interval.

**Exemple:**

1. Expresia `[Elevi].[Telefon] Is Null` returnează valoarea *True* doar în cazul în care câmpul *Telefon* al tabelului *Elevi* nu conține nicio valoare (se are în vedere înregistrarea curentă).
2. *Like "B\*ov"* specifică siruri de caractere care încep cu litera *B* și se termină cu combinația de litere *ov*. Prin urmare, expresia *"Belousov"* *Like "B\*ov"* returnează valoarea *True*.
3. *Like "[CK]#?"* specifică siruri din 3 caractere: primul este una dintre literele *C* sau *K*, al doilea – o cifră, al treilea – orice simbol.
4. *Like "\*[5ad-g]"* specifică siruri de caractere care se termină cu cifra 5 sau cu una dintre literele *a*, *d*, *e*, *f*, *g*.
5. *Like "\*[!ae]"* specifică siruri de caractere care nu se termină cu litera *a* sau cu litera *e*.
6. Expresia *"Duminica"* *In ("Luni"; "Marti"; "Miercuri"; "Joi"; "Vineri")* returnează valoarea *False*, iar expresia *4 In (2; 4; 7; 8)* – valoarea *True*.
7. Expresia *Between 2 And 10* este echivalentă cu expresia *>=2 And <=10*.

**Observație:** La scrierea expresiei condiției de validare în caseta proprietății *Validation Rule* nu se scrie identificatorul primului operand, acesta fiind considerat implicit identificatorul câmpului respectiv. Astfel, regula de validare *In ("Luni"; "Marti"; "Miercuri"; "Joi"; "Vineri")* va permite utilizatorului să scrie în câmp doar unul dintre cuvintele *Luni*, *Marti*, *Miercuri*, *Joi*, *Vineri*.

## Funcții Access

O funcție returnează o valoare prin numele ei. Access oferă peste 100 de funcții standarde pentru prelucrarea diferitelor tipuri de date: numerice, siruri de caractere, calendaristice etc. Cele mai uzuale sunt prezentate în următoarele tabele:

## Funcții pentru prelucrarea datelor calendaristice

Funcția	Rezultatul returnat
Date()	Data curentă
DateAdd( $T; N; D$ )	Data calendaristică care se obține adunând la data $D$ sau scăzând din ea $N$ (în cazul când $N$ este negativ) unități calendaristice de tip $T$ , unde $T$ poate fi "yyyy", "q", "m", "ww", "d", "h", semnificând respectiv ani, trimestre, luni, săptămâni, zile, ore
DateDiff( $T; D_1; D_2$ )	Diferența exprimată în unități calendaristice de tip $T$ dintre datele $D_1$ și $D_2$
Time()	Ora curentă
Now()	Data și ora curentă
Year( $D$ )	Anul scris cu 4 cifre corespunzător datei calendarisice $D$
Month( $D$ )	Numărul de ordine în an al lunii corespunzătoare datei calendarisice $D$
Day( $D$ )	Numărul de ordine în lună al zilei corespunzătoare datei calendarisice $D$
WeekDay( $D$ )	Numărul de ordine în săptămână al datei calendaristice $D$ (1 corespunde Duminică, 2 – zilei de Luni etc.)
Hour( $D$ )	Ora (număr întreg de la 0 la 23) corespunzătoare valorii calendaristice $D$

## Funcții de prelucrare a textului

Funcția	Rezultatul returnat
Asc( $C$ )	Codul ANSI al caracterului $C$
Chr( $N$ )	Caracterul al cărui cod ANSI este numărul $N$
InStr( $S_1; S_2$ )	Pozitia, începând cu care sirul $S_2$ se conține în sirul $S_1$
Mid( $S; N_1; N_2$ )	Subsirul sirului $S$ de lungime $N_2$ începând cu poziția $N_1$ . Parametrul $N_2$ poate să lipsească, ceea ce înseamnă că se va returna subsirul obținut prin înlăturarea primelor $N_1 - 1$ caractere ale sirului $S$
LCase( $S$ )	Sirul de caractere obținut din sirul $S$ prin transformarea literelor majuscule în litere mici

Funcția	Rezultatul returnat
UCase( $S$ )	Şirul de caractere obținut din şirul $S$ prin transformarea literelor mici în majuscule
Len( $S$ )	Numărul de caractere ale şirului $S$
LTrim( $S$ )	Şirul obținut din şirul $S$ după lichidarea spațiilor de debut
RTrim( $S$ )	Şirul obținut din şirul $S$ după lichidarea spațiilor de sfârșit
Trim( $S$ )	Şirul obținut din şirul $S$ după lichidarea spațiilor de debut și a celor de sfârșit
Left( $S; N$ )	Şirul format din primele $N$ caractere ale şirului $S$
Right( $S; N$ )	Şirul format din ultimele $N$ caractere ale şirului $S$
Str( $X$ )	Şirul format din simbolurile valorii $X$ în aceeași ordine
Val( $S$ )	Numărul obținut din simbolurile şirului $S$ în aceeași ordine (dacă acesta are formatul potrivit)

## Functii matematice

Funcția	Rezultatul returnat
Abs( $X$ )	Valoarea absolută a numărului $X$
Atn( $X$ )	Arctangenta (în radiani) a numărului $X$
Avg( $C$ )	Media aritmetică a valorilor câmpului $C$
Count( $C$ )	Numărul valorilor nevide ale câmpului $C$
Max( $C$ )	Valoarea maximală din câmpul $C$
Cos( $X$ )	Cosinusul numărului $X$
Exp( $X$ )	Valoarea $e^X$
Int( $X$ )	Partea întreagă a numărului $X$

Funcția	Rezultatul returnat
Log(X)	Logaritmul zecimal al numărului X
Rnd()	Un număr aleator cuprins între 0 și 1
Sgn(X)	0 dacă numărul X este pozitiv, -1 dacă numărul X este negativ
Sin(X)	Sinusul numărului X
Sqr(X)	Rădăcina pătrată a numărului X
Tan(X)	Tangenta numărului X

### Observații:

- În calitate de parametri ai funcțiilor pot fi identificatorii câmpurilor (evidenț, scriși între simbolurile [ și ]).
- Dacă parametrul funcției este o constantă calendaristică, atunci ea se scrie între simbolurile " și ".

### Exemple:

- DateAdd("d";-50; Date())* returnează data calendaristică care a fost cu 50 de zile în urmă.
- Weekday("27.09.1993")* returnează 2, deoarece pe 28 septembrie 1993 a fost luni. (Luni se consideră a doua zi din săptămână.)
- InStr("Informatica"; "forma")* returnează 3.
- LCase("INFORMATICA")* returnează textul "informatica".
- LTrim("forma")* returnează textul "forma".
- Sgn(- 20)* returnează valoarea -1.

## Întrebări și exerciții

1 **Explică!** Pentru ce se utilizează expresiile în Access?

2 **Aplică!** Deschideți baza de date *Liceu* și scrieți o regulă de validare pentru câmpul *Nr\_ore\_saptamana* al tabelului *Prof\_dis\_clasa* care să admită doar valori întregi strict mai mari ca 0.

3 **Exersează!** Ce valoare va returna expresia:

- Mid("Propoziție"; 3);*
- Mid("Calculator"; 1; 3);*
- Int(Rnd()\*50);*
- Month("15.11.1990");*
- Left("Tractor"; 5);*

- f)  $5 \ln ("4"; "5"; "6"; "7"; "8")$ ;
- g)  $\text{Val} ("25") - 25$ ;
- h) "R" Like "[TR]\*";
- i) "R" + Str(Date())?

4 **Aplică!** Scrieți o expresie care va returna:

- a) salariul mediu al profesorilor din tabelul *Profesori*;
- b) data calendaristică ce va fi peste 5 săptămâni după data curentă;
- c) a câțiva zile în an este data curentă;
- d) a câteva zile din săptămână a fost 1 ianuarie 2000;
- e) diferența de zile dintre 5 martie 2005 și 5 decembrie 2005.

5 **Învață să înveți!** Scrieți pentru câmpul *Gen\_prof* al tabelului *Profesori* o condiție de validare care ar admite în câmp doar valorile *M* sau *F*.

4.9

## Noțiuni generale despre interogări

Sistemele de gestiune a bazelor de date au fost concepute pentru a păstra informații și pentru a automatiza procesul de prelucrare a acestor informații. Chiar și pentru o bază de date cu câteva sute de înregistrări, căutarea manuală a datelor care satisfac anumite condiții este anevoieasă. Menționăm că există baze de date care conțin milioane de înregistrări. De exemplu, unii operatori de telefonie mobilă din Republica Moldova au peste 1 000 000 de abonați! La ultimele alegeri parlamentare din țară au participat peste 1,5 milioane de alegători. Deci, în cazul votării electronice, se vor prelucra peste 1,5 milioane de înregistrări ale unei baze de date!

Căutarea unor date poate implica consultarea câtorva tabele. De exemplu, pentru a afla ce discipline studiază un elev anume din baza de date *Liceu*, trebuie să examinăm tabelele *Elevi*, *Clase*, *Prof\_dis\_clasa* și *Discipline*. Să ne convingem!

- Consultați baza de date *Liceu* și determinați ce discipline studiază elevul *Dan Moraru*.

Exercițiul propus este doar un mic argument menit să demonstreze necesitatea cercetării și prelucrării automatizate a informațiilor unei baze de date.

Pentru selectarea rapidă din unul sau din mai multe tabele a seturilor de date care corespund unor condiții, dar și pentru actualizarea accelerată a înregistrărilor, sistemele de gestiune a bazelor de date utilizează *interrogări* – cereri de căutare și/sau de acțiune în conformitate cu aceste condiții.

**Interrogările sunt obiecte ale sistemelor de gestiune a bazelor de date care reprezintă adresări de căutare, analizare și/sau de modificare a datelor bazei.**

Mentionăm că în calitate de surse de date, în afară de înregistrările tabelelor bazei, o interogare poate folosi rezultatele altor interogări, executate anterior.

De exemplu, pentru a afișa lista profesorilor cu salariul maximal, se vor crea două interogări: prima interogare va găsi valoarea *Max* a salariului maximal, iar a doua – va selecta din tabelul *Profesori* înregistrările cu valorile din câmpul *Salariu*, egale cu *Max* (adică egale cu rezultatul primei interogări).

Interogările îndeosebi se utilizează pentru:

- vizualizarea unui subset de înregistrări dintr-un tabel, fără a-l deschide;
- afișarea într-un singur tabel a informațiilor din câteva tabele;
- actualizarea datelor tabelelor (modificări, adăugări, excluderi de date);
- efectuarea calculelor asupra valorilor câmpurilor și obținerea informațiilor noi;
- crearea totalurilor, valorilor medii etc.

În funcție de tipul acțiunii și de rezultate, interogările se clasifică în cele:

- a) *de selecție*;
- b) *de excludere a unor înregistrări*;
- c) *de modificare a unor înregistrări*;
- d) *de creare a câmpurilor rezultante*;
- e) *de grupare și totalizare*;
- f) *încrucișate*.

**Interogările de selecție** sunt cereri formulate în baza unor condiții logice. Ele selectează un subset de date din unul sau mai multe tabele legate între ele. De exemplu, căutarea elevilor născuți până la 10 ianuarie 2006, afișarea elevilor clasei a 10-a B sunt interogări de selecție.

**Interogările de excludere a unor înregistrări** reprezintă cereri de eliminare dintr-un tabel a tuturor înregistrărilor care satisfac criteriile logice specificate. De exemplu, cererea de ștergere din tabelul *Elevi* al bazei de date *Liceu* a informațiilor despre elevii și elevele claselor a 12-a (în legătură cu absolvirea liceului) reprezintă o astfel de interogare.

**Interogările de modificare a unor înregistrări** schimbă valorile unui câmp al tabelului după același algoritm. De exemplu, mărirea cu 50% a valorilor din câmpul *Salariu* al tabelului *Profesori* poate fi realizată cu ajutorul unei interogări de modificare.

Uneori se pot solicita informații care trebuie afișate în câmpuri noi. De exemplu, vârsta elevilor se va prezenta într-un câmp aparte. Cu acest scop, se va utiliza o **interogare de creare a câmpurilor rezultante**.

**Interogările de grupare și totalizare** se folosesc pentru sumarea datelor câmpurilor, obținerea valorilor medii, a celor minime sau maxime etc. De exemplu, calcularea numărului total de ore pe săptămână realizate în fiecare clasă a bazei de date *Liceu* se va face cu ajutorul unei interogări de grupare și totalizare.

**Interogările încrucișate** sunt destinate prezentării compacte a informațiilor sub formă de tabel. De exemplu, informația despre numărul săptămânal de ore rezervat fiecărei discipline în fiecare clasă poate fi afișată cu ajutorul unei interogări încrucișate sub forma următorului tabel:

Anul_de_studii	Nume_clasa	Biologia	Chimia	...
10	A	2	3	...
10	D	1	1	...
11	A	3	2	...
12	D	1	1	...
...	...	...	...	...

Aşa cum rezultatele interogărilor depind de informaţiile din tabele, modificările realizate asupra datelor din tabele vor atrage după sine modificarea rezultatelor interogărilor (evident după executarea lor repetată).

La rândul său, în cazul interogărilor a)-d), modificarea de către utilizator a rezultatelor interogării poate duce la schimbarea conținutului tabelelor. Din aceste considerente, rezultatul interogărilor a)-d) se numeşte **set dinamic de date**.

Setul dinamic de date nu este constant, deci nu se memorează. El există doar pe parcursul execuţiei interogării.

#### Pentru a crea o interogare nouă:

**1. Executăm un clic pe butonul *Query Design*** (  ) din meniu *Create*. Automat apare şi meniu *Query Design*.

**2. Apare fereastra interogării şi fereastra de dialog *Show Table* (fig. 4.11).**

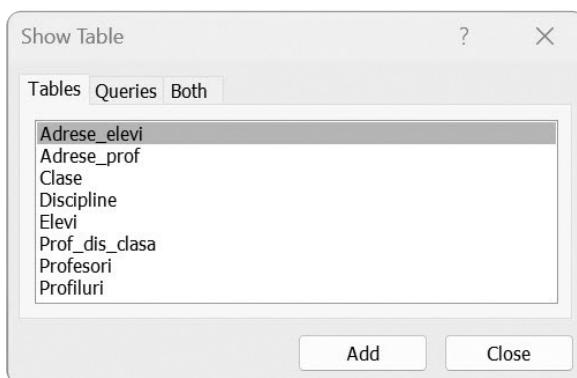


Fig. 4.11. Fereastra de dialog *Show Table*  
(Afişează tabelul)

**Selectăm pe rând tabelele** necesare apăsând de fiecare dată butonul *Add*. Tabelele alese (mai exact câmpurile lor) apar în interiorul ferestrei interogării împreună cu reprezentarea grafică a relațiilor dintre ele (fig. 4.12).

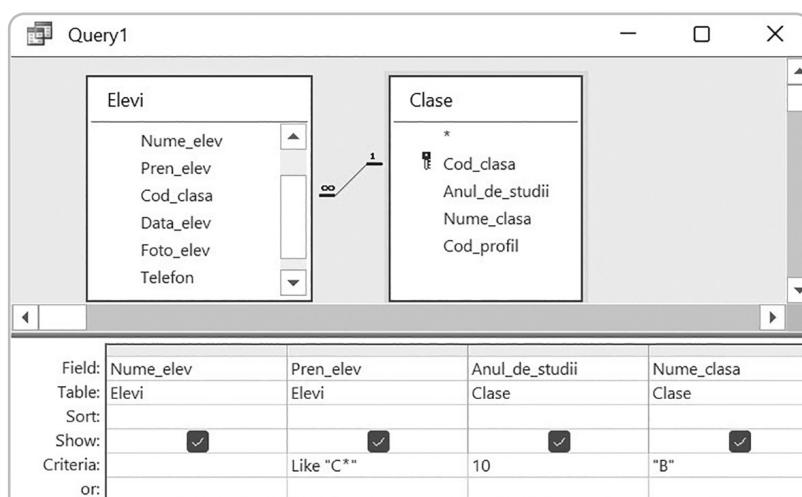


Fig. 4.12. Fereastra de creare a unei interogări de selectare

3. Fereastra unei interogări este divizată în două zone. Zona superioară afișează denumirile câmpurilor tabelelor. Zona inferioară afișează un **formular QBE**\* în care programatorul poate folosi exemple de declarații parțiale sau expresii pentru a crea o interogare. Alegem câmpurile care vor fi utilizate pentru scrierea criteriului de selecție și câmpurile ale căror valori vor fi afișate. Adăugarea unui câmp se face prin executarea unui dublu clic pe denumirea lui sau prin selectarea și „tragerea” acestui câmp cu ajutorul mouse-ului. Ordinea câmpurilor afișate în formular poate fi schimbată în același mod ca și ordinea coloanelor unui tabel afișate în regimul *Datasheet View*.
4. Pentru fiecare câmp adăugat în formularul QBE, în afară de câmpul și tabelul care conține acest câmp (afișate în rândurile *Field* și *Table*), se pot preciza:
- modul de sortare a înregistrărilor după acest câmp (rândul *Sort*), alegând una dintre valorile *Ascending* (crescător) sau *Descending* (descrescător);
  - afișarea sau ascunderea valorilor câmpului la executarea interogării (rândul *Show*);
  - o condiție de selectare pe care o vor respecta valorile afișate (rândurile *Criteria* și *or*).
5. Putem **afișa datele specificate** de interogare până la salvarea ei prin selectarea comenzii *Datasheet View* din meniul *View* (din meniul *Query Design*).
6. **Salvăm interogarea** (instrumentul *Save* sau comanda cu același nume din meniul *File*).

**Exemplu:** În figura 4.12 este prezentată în regim de proiectare o interogare care cere afișarea listei elevilor clasei a X-a B, al căror prenume începe cu litera C. În figura 4.13 este prezentat rezultatul interogării – setul dinamic cu datele selectate –, până la salvarea interogării propriu-zise.

The screenshot shows a Microsoft Access query window titled "Query1". It displays a table with four columns: "Nume\_ele", "Pren\_elev", "Anul\_de\_stu", and "Nume\_clas". The data consists of three rows: Bujor (Călin, 10 B), Cozariuc (Cătălina, 10 B), and Mihalachi (Cristina, 10 B). The bottom of the window shows navigation buttons for "Record" (including back, forward, and search), a "No Filter" button, and a "Search" button.

Fig. 4.13. Interogarea în curs de proiectare, afișată în regimul *Datasheet View* (Vizualizare în regim Foaie de date)

## Întrebări și exerciții

- 1 **Explică!** Cu ce scop se utilizează interogările?
- 2 **Sistematizează-ți cunoștințele!** Caracterizați principalele tipuri de interogări.
- 3 **Explică!** Descrieți algoritmul de creare a unei interogări.
- 4 **Explică!** De ce rezultatul unor interogări se numește *set dinamic de date*?
- 5 **Aplică!** Stabiliti tipul următoarelor interogări:
  - aflarea numărului de profesori de gen masculin și a numărului celor de gen feminin;
  - determinarea salariului mediu al profesorilor din baza de date *Liceu*;
  - afișarea listei fetelor din clasele a XI-a;
  - determinarea numărului de ore realizate săptămânal de fiecare profesor;
  - căutarea profesorilor de gen feminin care predau în clasele a X-a.

6

**Proiectează!**

Examinați baza de date Liceu și formulați două cereri de interogări:

- a) de selecție;
- b) de excludere a unor înregistrări;
- c) de modificare a unor înregistrări;
- d) de creare a câmpurilor rezultante;
- e) de grupare și totalizare;
- f) încrucișate.

4.10

## Interogări de selecție

Pornind de la faptul că interogările de selecție sunt cele mai des utilizabile, Access stabilește implicit acest tip pentru toate interogările nou-create. După cum vom vedea ulterior, utilizatorul trebuie să întreprindă acțiuni suplimentare pentru a schimba tipul interogării noi.

Așadar, pentru a crea o interogare de selecție, vom alege tabelele și câmpurile necesare conform algoritmului descris în tema precedentă.

### Criterii de selecție

Un moment important în procesul de elaborare a unei astfel de interogări este scrierea **criteriului de selecție**.

Dacă criteriul se formulează pentru un singur câmp, atunci expresia logică, care va controla afișarea datelor, se scrie în celula *Criteria* a acestui câmp. Menționăm că operatorul *Like* este inserat automat de Access în cazul în care utilizatorul scrie un şablon de restricționare a datelor. În particular, în celula *Criteria* se poate scrie o constantă de tip compatibil cu tipul valorilor câmpului respectiv.

**Exemplu:** Pentru a afișa lista elevilor cu prenumele Ion, este suficient să se scrie în celula *Criteria* a câmpului *Pren\_elev* (din tabelul *Elevi*) sirul de caractere „Ion”.

Deoarece criteriile de selecție sunt expresii logice Access, pentru scrierea lor pot fi utilizate funcțiile și operatorii studiați în capitolul precedent, inclusiv cei *logici*.

În același timp, formularul QBE oferă ajutor în crearea criteriilor compuse, formate din câteva condiții și din operatorii AND și/sau OR. Astfel:

- a) pentru un câmp pot fi definite mai multe condiții de selecție legate între ele cu operatorul OR: prima se scrie în rândul *Criteria*, iar celelalte – mai jos, câte una în fiecare celulă;
- b) două sau mai multe condiții din rândul *Criteria* se consideră legate între ele cu operatorul AND.

**Exemple:**

1. Dacă în formularul interogării anterioare mai jos de valoarea „Ion” (în celula *or*) se va scrie „Vasile”, atunci interogarea va afișa lista elevilor cu prenumele Ion sau Vasile. În cazul completării celulei *Criteria* din câmpul *Nume\_elev* cu condiția „B”, atunci interogarea va afișa lista elevilor cu prenumele Ion și al căror nume începe cu litera „B”.
2. Interogarea din figura 4.14 va selecta elevii și elevele din clasa a XI-a B născuți în luna ianuarie.
3. Interogarea din figura 4.15 va selecta elevii și elevele din clasele 10–11 cu profil real. Evident, rezultatul va fi același dacă vom completa doar rândul *Criteria*, substituind valoarea 10 cu expresia 10 OR 11.
4. Interogarea din figura 4.16 va selecta profesorii ce au un salariu lunar mai mare decât 9 000 de lei și mai mic sau egal cu 10 000 de lei.

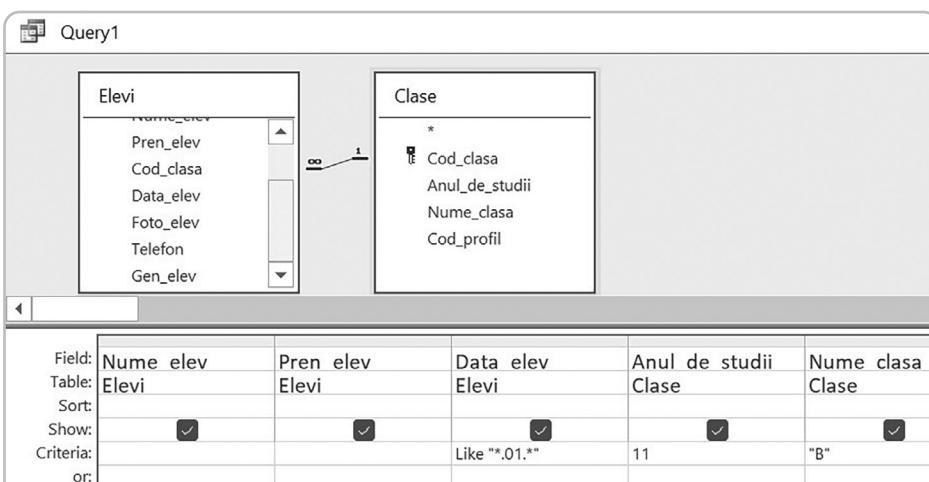


Fig. 4.14. Interogarea de selectare a elevilor clasei a XI-a B născuți în luna ianuarie

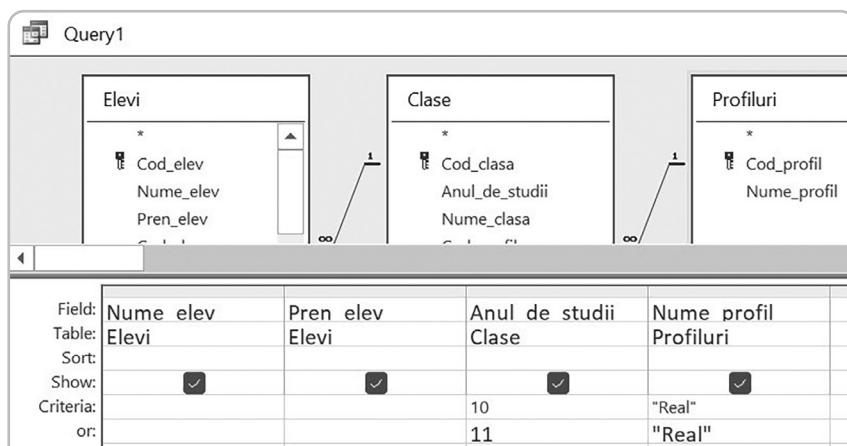


Fig. 4.15. Interogarea de selectare a elevilor din clasele X–XI cu profil real

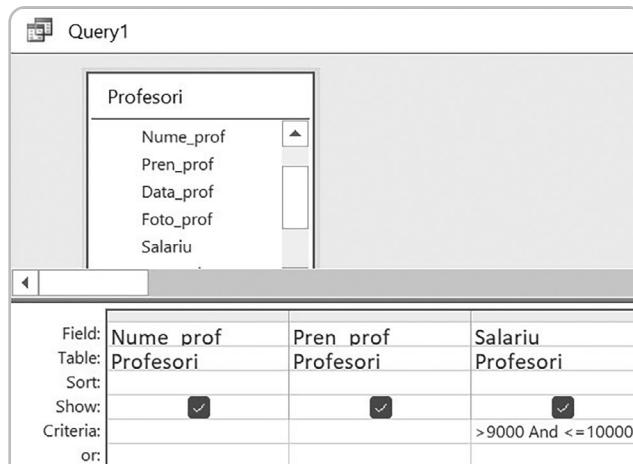


Fig. 4.16. Interogarea de selectare a profesorilor ce au un salariu lunar mai mare decât 9 000 de lei și mai mic sau egal cu 10 000 de lei

## Interogări cu parametri

Evident, este imposibil de a prezice toate cererile de selecție de care vor avea nevoie utilizatorii. Mai mult chiar, unele interogări se pot deosebi între ele doar prin unele valori din formularul QBE. De exemplu, interogările care vor afișa lista disciplinelor studiate în clasa a X-a B, respectiv, în clasa a XI-a B, se vor deosebi doar prin valorile câmpului *Anul\_de\_studii*.

În astfel de situații se poate crea o singură interogare, unde în loc de valoare, în câmpul corespunzător, se va scrie *un parametru*.

Un sir de caractere încadrat între simbolurile [ și ], scris într-o celulă a rândului *Criteria* este interpretat de Access drept **parametru**.

Pentru fiecare parametru, la executarea interogării, mai întâi va apărea o fereastră de dialog în care utilizatorul va scrie valoarea parametrului (deci o valoare a câmpului pentru care a fost creat parametrul), apoi se vor afișa înregistrările ale căror valori din câmpul parametrului coincid cu valoarea scrisă de utilizator.

De regulă, sirul de caractere ce definește parametrul este un text explicativ, care sugerează utilizatorului ce tip de valoare trebuie să scrie în fereastra de dialog.

În mod implicit parametrul se consideră de tip *Text*. Pentru a modifica tipul parametrului, se va apela butonul *Parameters* din meniul *Query Design*. Va apărea fereastra *Query Parameters*, în care se va scrie fiecare parametru și tipul lui.

**Exemplu:** În figura 4.17 este reprezentată fereastra unei interogări de selecție cu trei parametri: primul pentru câmpul *Anul\_de\_studii*, al doilea – pentru câmpul *Nume\_clasa*, iar al treilea (de tip *Number*) – pentru câmpul *Nr\_ore\_saptamana*. Observăm că prin intermediul acestei interogări se poate afla lista disciplinelor studiate de elevii și elevele clasei indicate de utilizator, al căror număr de ore pe săptămână este de asemenea specificat de utilizator.

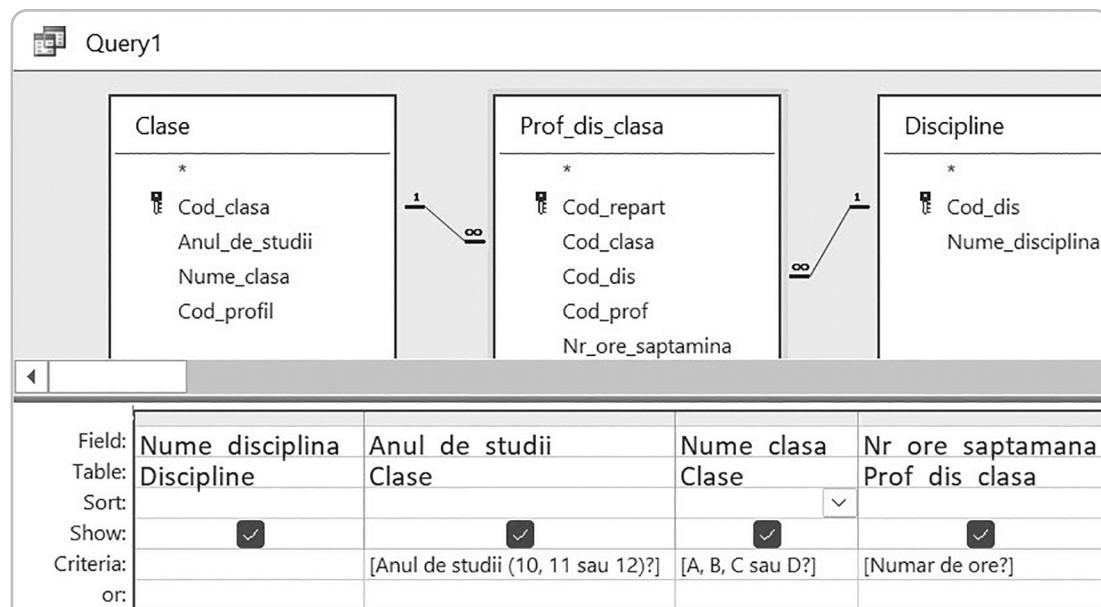


Fig. 4.17. Interogare de selecție cu trei parametri

La executarea interogării din *figura 4.17* vor apărea, pe rând, trei ferestre în care utilizatorul va preciza anul de studii, numele clasei (A, B, C sau D) și, respectiv, numărul săptămânal de ore.

## Întrebări și exerciții

- 1 Sistematizează-ți cunoștințele!** Ce constituie interogările de selecție?
- 2 Explică!** Care este rolul criteriului de selecție într-o interogare?
- 3 Sistematizează-ți cunoștințele!** Cu ce scop se folosesc parametrii într-o interogare?
- 4 Aplică!** Deschideți baza de date *Liceu*. Creați o interogare de selecție ce va afișa lista:
  - a) fetelor bazei;
  - b) dirigintilor;
  - c) profesorilor născuți până la 12 februarie 1970;
  - d) profesoarelor născute în luna martie;
  - e) băieților în ordine crescătoare a vîrstei lor;
  - f) elevilor care nu au telefon;
  - g) profesorilor de gen masculin care nu au împlinit 50 de ani;
  - h) elevilor născuți vara;
  - i) profesorilor al căror număr de telefon începe cu 022 48;
  - j) elevilor care nu locuiesc în Chișinău;
  - k) profesorilor al căror prenume începe cu litera A sau cu litera E;
  - l) elevilor care nu învață Filosofia;
  - m) elevilor care studiază Matematica 5 ore pe săptămână.
- 5 Aplică!** Deschideți baza de date *Liceu*. Creați o interogare de selecție cu parametru ce va afișa lista:
  - a) disciplinelor studiate de elevul specificat de utilizator;
  - b) profesorilor care predau în clasa specificată de utilizator;
  - c) profesorilor care locuiesc în localitatea specificată de utilizator;
  - d) elevilor care studiază disciplina specificată de utilizator;
  - e) profesorilor care predau disciplina specificată de utilizator.
- 6 Învață să înveți!** Formulați și creați 5 interogări de selecție pentru baza de date *Liceu*.
- 7 Învață să înveți!** Pentru baza de date *Liceu* formulați și creați 5 interogări de selecție cu parametru.

## 4.11 Interogări de acțiune

Interogările de acțiune se utilizează pentru a crea tabele noi în baza informațiilor din tabelele existente și/sau pentru a realiza modificări în aceste tabele. În fereastra bazei de date numele acestor interogări este precedat de semnul exclamării.

**Atenție!** Interogările de acțiune (cu excepția celor care generează tabele) modifică conținutul tabelelor.

### Interogări care generează tabele

Interogările de selecție extrag date din tabele și le afișează doar la executarea interogării. Rezultatul unei astfel de interogări, denumit set dinamic al interogării, nu se păstrează (eventual, într-un tabel). Pentru a păstra rezultatele extragerii, interogarea de selecție poate fi transformată într-o interogare care va genera un tabel nou, ce va conține rezultatele respective.

Pentru exemplificare, să elaborăm o interogare care va genera un tabel cu date despre elevii și elevele claselor a X-a.

1. Elaborăm interogarea de selecție corespunzătoare (fig. 4.18).
2. Din meniul *Query Design* alegem *Make-Table*. Apare fereastra *Make Table*, unde precizăm numele tabelului nou (de exemplu, *Clasele\_10*) și numele bazei de date în care va fi păstrat tabelul. Implicit, tabelul va fi creat în baza de date curentă.
3. Scriem numele tabelului, confirmăm prin apăsarea butonului OK, apoi salvăm interogarea.
4. Pentru a obține tabelul *Clasele\_10*, executăm interogarea recentă (apăsăm butonul *Run* din meniul *Query Design*). Apare o fereastră cu mesaj de avertizare, în care confirmăm intenția de creare a tabelului prin apăsarea butonului Yes.

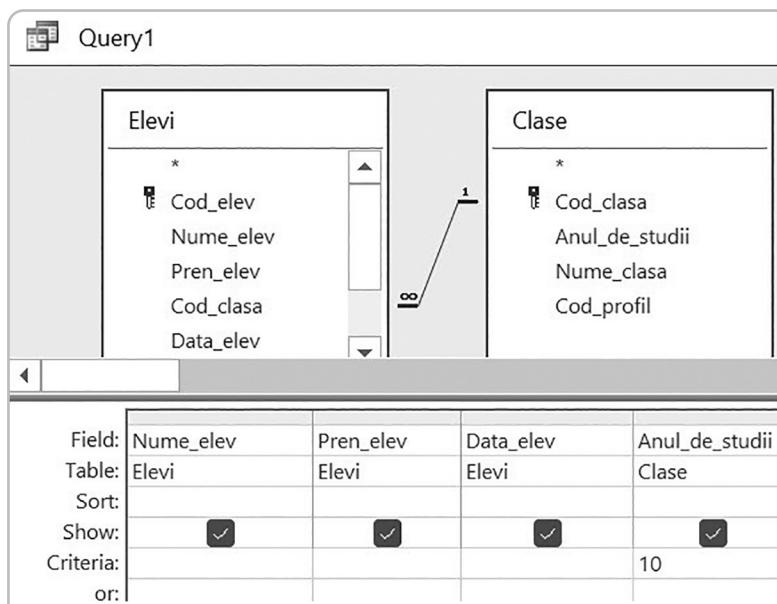


Fig. 4.18. Interogarea de selectare a elevilor din clasa a X-a

## Interogări de excludere a unor înregistrări

Deseori apare necesitatea eliminării unor înregistrări din tabele. De exemplu, în cazul bazei de date *Liceu* să admitem că trebuie să excludem datele despre elevii și elevele claselor a XII-a (în legătură cu absolvirea liceului).

1. Așa cum între tabelele *Clase* și *Elevi* există relația **unu la mulți**, având caracteristica *Excluderea în cascadă a înregistrărilor* (*Cascade Delete Related Records*), este suficient să ștergem înregistrările din tabelul *Clase* care au valoarea câmpului *Anul\_de\_studii* egală cu 12. Învers nu este corect, deoarece tabelul *Elevi* este subordonat tabelului *Clase*.
2. Elaborăm o interogare care afișează lista claselor. Din meniul *Query Design* alegem *Delete*. În formularul QBE, în locul rândurilor *Show* și *Sort*, apare rândul *Delete*. În celula *Criteria* a câmpului *Anul\_de\_studii* scriem 12 (fig. 4.19).

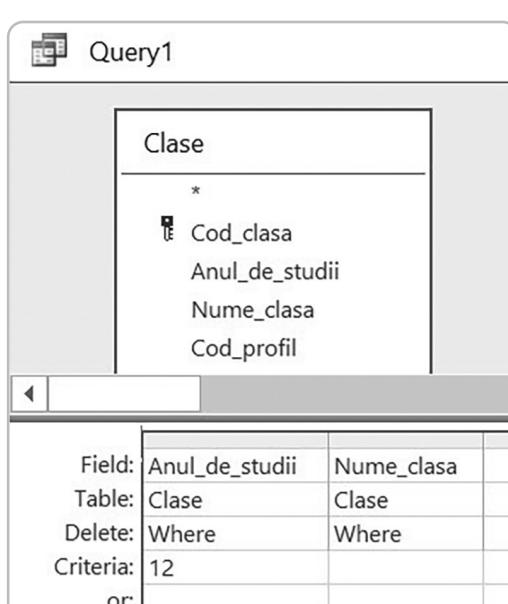


Fig. 4.19. Interogare de ștergere a înregistrărilor (Delete Query)

3. Salvăm interogarea. Ștergerea înregistrărilor se va realiza după executarea interogării.

**Atenție!** Așa cum înregistrările sterse nu pot fi restabile, se recomandă de afișat rezultatul interogării de selecție înainte ca aceasta să fie transformată în una de excludere a înregistrărilor. Astfel se va verifica corectitudinea acțiunii interogării.

## Interogări de modificare a unor înregistrări

Dacă valorile mai multor înregistrări ale unui câmp pot fi transformate după același algoritm, atunci pentru automatizarea actualizărilor se va folosi o interogare de modificare.

Pentru exemplificare, să elaborăm o interogare care va mări cu 50% valorile din câmpul *Salariu* al tabelului *Profesori*.

1. Creăm o interogare de selecție care afișează valorile câmpului *Salariu*.
2. Din meniul *Query Design* alegem *Update*. În formularul QBE în locul rândurilor *Sort* și *Show* apare rândul *Update To*. Scriem expresia  $[Salariu]*1,5$  în celula obținută la intersecția rândului *Update To* și a câmpului *Salariu* (fig. 4.20).

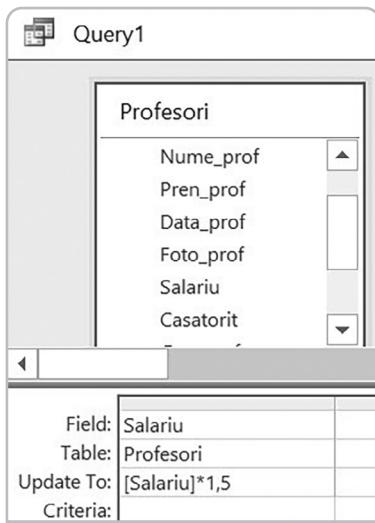


Fig. 4.20. Interogare de actualizare (*Update Query*) a salariilor profesorilor

3. Salvăm interogarea. Modificarea înregistrărilor se va realiza după executarea interogării.

**Atenție!** De obicei, interogările de modificare trebuie lansate în execuție doar o singură dată. În caz contrar, la a doua lansare în execuție, noile valori, obținute după prima lansare, vor fi modificate din nou și.a.m.d. Astfel, dacă vom executa de 2 ori interogarea din exemplul precedent, salariile profesorilor vor fi mărite nu cu 50%, după cum se intenționa, ci cu 125%.

### Interogări care adaugă înregistrări în tabele existente

Uneori trebuie să adăugăm într-un tabel înregistrări din alt tabel. De exemplu, fie tabelul *Elevi1* care conține date despre elevii și elevele născuți în luna mai. Dacă dorim să adăugăm în acest tabel înregistrările elevilor din tabelul *Elevi* născuți în luna iunie, vom proceda astfel:

1. Creăm o interogare de selecție care va afișa lista elevilor născuți în luna iunie.
2. Din meniul *Query Design* alegem *Append*. Apare fereastra *Append*. Din lista derulantă *Table Name* selectăm *Elevi1*.
3. Salvăm, apoi executăm interogarea.

## Întrebări și exerciții

- 1 Explică!** Cu ce scop se utilizează interogările de acțiune?
- 2 Sistematizează-ți cunoștințele!** Descrieți algoritmul de creare a unei interogări:
  - a) care generează tabele;
  - b) de modificare a unor înregistrări;
  - c) de excludere a unor înregistrări;
  - d) care adaugă înregistrări în tabelele existente.
- 3 Elaborează!** Se consideră baza de date *Liceu*. Creați o interogare ce va genera tabelul:
  - a) T1 cu date despre profesorii de gen feminin;
  - b) T2 cu date despre elevii și elevele născuți în luna mai;
  - c) T3 cu date despre dirigintii de clase;
  - d) T4 cu date despre profesorii care predau matematica sau chimia;
  - e) T5 cu date despre elevii și elevele care nu locuiesc în Chișinău.
- 4 Elaborează!** Se consideră baza de date *Liceu*. Elaborați o interogare ce șterge din tabelul:
  - a) T1 datele despre profesorii născuți vara;
  - b) T2 datele despre elevii și elevele clasei a XI-a;
  - c) T3 datele despre dirigintii claselor cu profil real;
  - d) T4 datele despre profesorii care nu predau chimia;
  - e) T5 datele despre elevii și elevele care locuiesc în Cricova.
- 5 Învață să înveți!** Se consideră baza de date *Liceu*. Alcătuți o interogare ce va:
  - a) mări salariul profesorilor cu 500 de lei;
  - b) micșora cu 300 de lei salariul profesorilor care predau doar o disciplină;
  - c) mări cu 25% salariul profesorilor născuți până la 01.01.1960.
- 6 Învață să înveți!** Se consideră baza de date *Liceu*. Creați o interogare ce va adăuga în tabelul:
  - a) T1 date despre profesorii de gen masculin care predau matematica;
  - b) T2 date despre elevii și elevele născuți iarna;
  - c) T3 date despre profesorii care nu sunt diriginti și locuiesc în Chișinău;
  - d) T4 date despre profesorii care predau limba străină;
  - e) T5 date despre elevii și elevele din Chișinău care învață în clasa a X-a.

4.12

## Interogări de totalizare

### Interogări de creare a câmpurilor rezultante (calculate)

În capitolele precedente am menționat că la proiectarea entităților unei baze de date relationale se vor exclude câmpurile ale căror valori pot fi obținute din câmpurile rămase.

Din aceste considerente, în tabelul *Elevi* nu a fost inclus câmpul *Varsta*. Valorile acestui câmp depind de valorile câmpului *Data\_elev*. Să elaborăm o interogare care va afișa într-un câmp nou vârstele elevilor bazei de date *Liceu*. Interogarea nu va afecta structura și datele tabelului *Elevi*.

- Creăm o interogare de selecție pe baza tabelului *Elevi*. În primele două coloane ale formularului QBE includem câmpurile *Nume\_elev*, *Pren\_elev*, iar în locul denumirii coloanei a treia scriem expresia *Varsta*: *DateDiff("yyyy";[Data\_elev];Date())*. Menționăm că *Varsta* este identifierul câmpului nou, iar funcția *DateDiff(T; D<sub>1</sub>; D<sub>2</sub>)* returnează diferența exprimată în unități calendaristice *T* dintre datele *D<sub>1</sub>* și *D<sub>2</sub>* (a se vedea tema *Funcții Access*). Observăm că:
  - T* este egal cu "yyyy", deci exprimă ani;
  - D<sub>1</sub>* este *[Data\_elev]*, adică data nașterii elevului;
  - D<sub>2</sub>* este *Date()*, adică data curentă.
- Salvăm interogarea. Câmpul *Varsta* este dinamic. El există atât timp cât sunt afișate rezultatele interogării.

## Interogări de grupare și totalizare

Pentru a obține rezultate bazate pe înregistrările unui sau ale mai multor tabele, se vor utiliza *interogări de grupare și totalizare*.

- Să definim o interogare care va afișa numărul total de elevi ai fiecărei clase din baza de date *Liceu*.
- Creăm o interogare de selecție pe baza tabelelor *Clase* și *Elevi* în care includem câmpurile *Anul\_de\_studii*, *Nume\_clasa* și *Cod\_elev*.
  - Executăm un clic pe butonul *Totals* ( $\Sigma$ ) de pe bara de instrumente. În formularul QBE apare rândul *Total* (fig. 4.21). Completăm celulele acestui rând:
    - în primele două coloane din listele derulante ale celulelor selectăm valoarea *Group By* (deoarece grupăm înregistrările după anul de studii și numele clasei), iar
    - în coloana a treia selectăm funcția *Count* (deoarece calculăm numărul de înregistrări ale câmpului *Cod\_elev*).

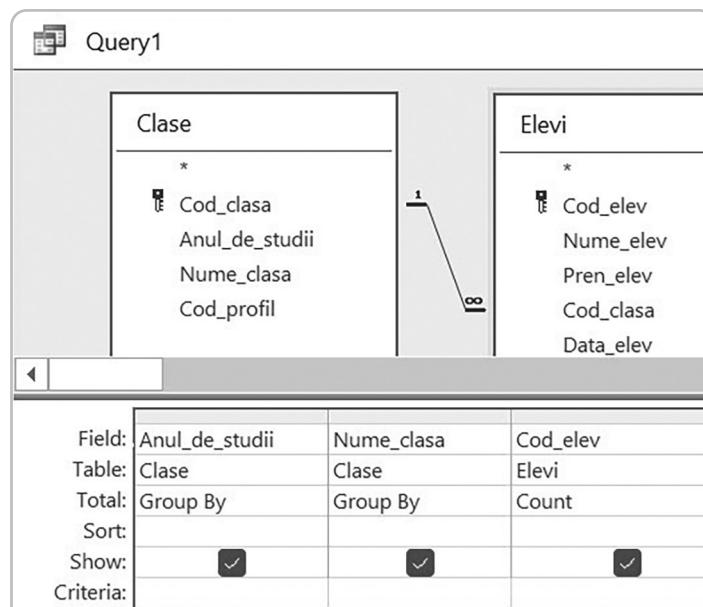


Fig. 4.21. Interogare de grupare și totalizare

3. Salvăm și executăm interogarea. Rezultatul interogării este prezentat în figura 4.22.

Anul_de_studii	Nume_clasa	CountOfCod_elev
10	A	29
10	B	18
10	C	37
10	D	31
11	A	33
11	B	26
11	C	22
11	D	23
12	A	22
12	B	24
12	C	20

Fig. 4.22. Setul dinamic de date generat de o interogare de grupare și totalizare

#### Observații:

1. Listele derulante ale celulelor rândului *Total* oferă diverse *funcții globale* (funcții aplicabile asupra grupurilor de celule de date) pentru obținerea totalizărilor: *Sum*, *Max*, *Min*, *Avg*, *First* etc.
2. În interogări de grupare și totalizare, de asemenea, se pot formula criterii de selecție. De exemplu, dacă pentru câmpul *Cod\_elev* al interogării precedente se va scrie condiția  $>25$ , atunci interogarea va afișa datele doar despre clasele al căror număr total de elevi este mai mare decât 25.

## Interogări încrucișate

Interogările încrucișate sunt interogări de totalizare care permit utilizatorului să stabilească exact modul în care vor fi afișate rezultatele sub formă tabelară. Astfel de interogări sunt recomandate în cazul unei cantități mari de date totalizatoare. La crearea unei interogări încrucișate se va ține cont de următoarele restricții:

- a) denumirile rândurilor tabelului-rezultat pot fi valori din unul sau mai multe câmpuri;
- b) denumirile coloanelor tabelului-rezultat pot fi valori doar ale unui singur câmp;
- c) valorile celorlalte celule ale tabelului-rezultat sunt calculate cu ajutorul unei funcții globale;
- d) înregistrările din rezultat nu pot fi ordonate după câmpurile celulelor calculate.

Să elaborăm o interogare care va afișa numărul total de ore rezervat fiecărei discipline din fiecare clasă a bazei de date *Liceu*.

1. Creăm o interogare de selecție pe baza tabelelor *Clase*, *Discipline* și *Prof\_dis\_clasa* în care includem câmpurile *Anul\_de\_studii*, *Nume\_clasa*, *Nume\_disciplina* și *Nr\_ore\_saptamana*.

2. Din meniul *Query Design* alegem *Crosstab*. În formularul QBE apar rândurile *Total* și *Crosstab* (fig. 4.23).

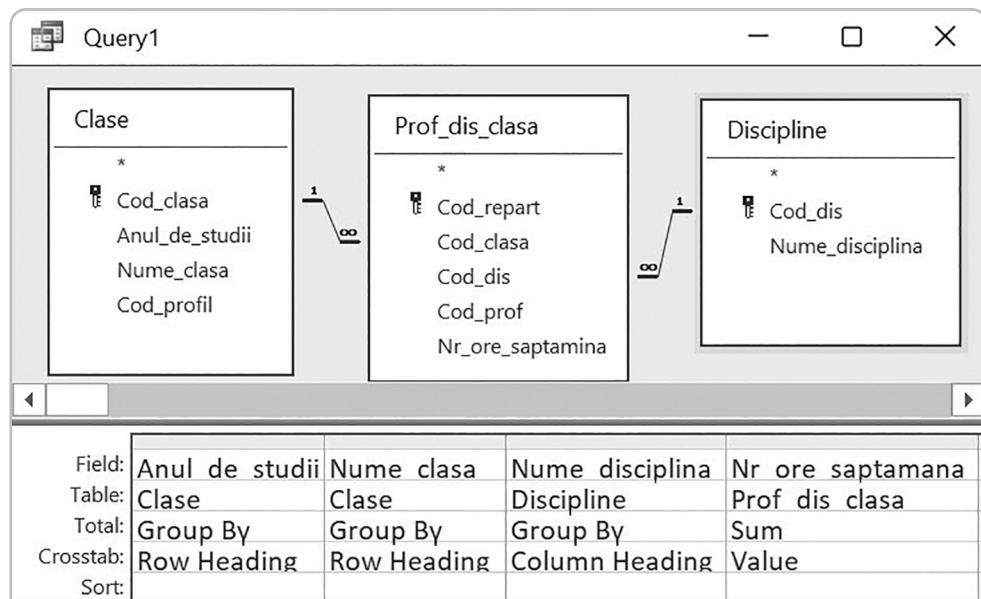


Fig. 4.23. Interogare încruzișată

3. Completăm celulele rândurilor *Total* și *Crosstab* ca în figura 4.23. Astfel, valorile câmpurilor *Anul\_de\_studii* și *Nume\_clasa* vor fi denumiri ale rândurilor tabelului-resultat, valorile câmpului *Nume\_disciplina* – denumiri ale coloanelor tabelului-resultat, iar valorile câmpului *Nr\_ore\_saptamana* vor fi sumate pentru a completa celelalte celule ale tabelului-resultat (fig. 4.24).

Anul_de_studi	Nume_clas	Bilologiu	Chimia	Educația civi	Ec
10 A		2	3	1	
10 B		2	3	1	
10 C		1	1	1	
10 D		1	1	1	
11 A		3	2	1	
11 B		3	2	1	
11 C		1	1	1	
11 D		1	1	1	
12 A		3	3	1	
12 B		3	3	1	
12 C		1	1	1	
12 D		1	1	1	

Fig. 4.24. Setul dinamic de date generat de o interogare încruzișată

## Întrebări și exerciții

**1 Explică!** Ce reprezintă interogările:

- a) de creare a câmpurilor rezultante;
- b) de grupare și totalizare;
- c) încruzișate?

**2 Sistematizează-ți cunoștințele!** Descrieți algoritmul de elaborare a unei interogări:

- a) de creare a câmpurilor rezultante;
- b) de grupare și totalizare;
- c) încruzișate.

**3 Aplică!** Se consideră baza de date *Liceu*. Aflați cu ajutorul unei interogări:

- a) vârsta elevilor exprimată în zile;
- b) numărul de profesori de gen masculin și numărul celor de gen feminin;
- c) numărul de clase la fiecare dintre profilurile *real* și *umanist*;
- d) salariul mediu lunar al profesorilor;
- e) numărul de elevi din fiecare localitate;
- f) numărul de ore realizate săptămânal de fiecare profesor;
- g) numărul de ore realizate săptămânal de fiecare clasă;
- h) valoarea salariului minimal;
- i) profesorii care au salariul maximal;
- j) numărul de profesori care au salariul mai mic decât cel mediu.

**4 Învață să înveți!** Se consideră baza de date *Liceu*. Elaborați și lansați în execuție trei interogări:

- a) de creare a câmpurilor rezultante;
- b) de grupare și totalizare;
- c) încruzișate.

4.13

## Formulare

**Formularele** sunt obiecte ale bazelor de date Access, proiectate special pentru a crea o interfață care facilitează introducerea, editarea și afișarea înregistrărilor din tabele și interogări.

Formularele măresc viteza și minimizează erorile de introducere a datelor. În același timp, ele permit vizualizarea datelor într-un format mai atractiv decât cel al regimului *Datasheet View*.

Cu ajutorul formularelor pot fi validate intrări de date pe baza informațiilor din alte tabele (decât cel la care se lucrează), pot fi create *subformulare* (formulare conținute în alte formule), *casete de căutare* (pentru accesarea rapidă a înregistrării), *liste de opțiuni* etc. Componentele unui formular se numesc **elemente de control** sau **obiecte de control**.

## Crearea unui formular cu ajutorul programului de asistență

Să creăm un formular care va permite editarea datelor din tabelul *Elevi* al bazei de date *Liceu*.

1. În meniul *Create* se află patru butoane pentru crearea formularelor:

- *Form* (crearea formularului pentru editarea înregistrărilor tabelului selectat. Este activ doar dacă este selectat un tabel);
- *Form Design* (crearea formularului în regim de proiectare);
- *Blank Form* (crearea unui formular vid, adică fără butoane de control casete etc.);
- *Form Wizard* (crearea formularului cu ajutorul unui program de asistență).

Selectăm ultima opțiune (*Form Wizard*).

2. Apare fereastra *Form Wizard* (fig. 4.25). Din lista combinată *Table/Queries* selectăm valoarea *Table: Elevi*. Astfel, în lista *Available Fields* apar identificatorii câmpurilor tabelului *Elevi*.

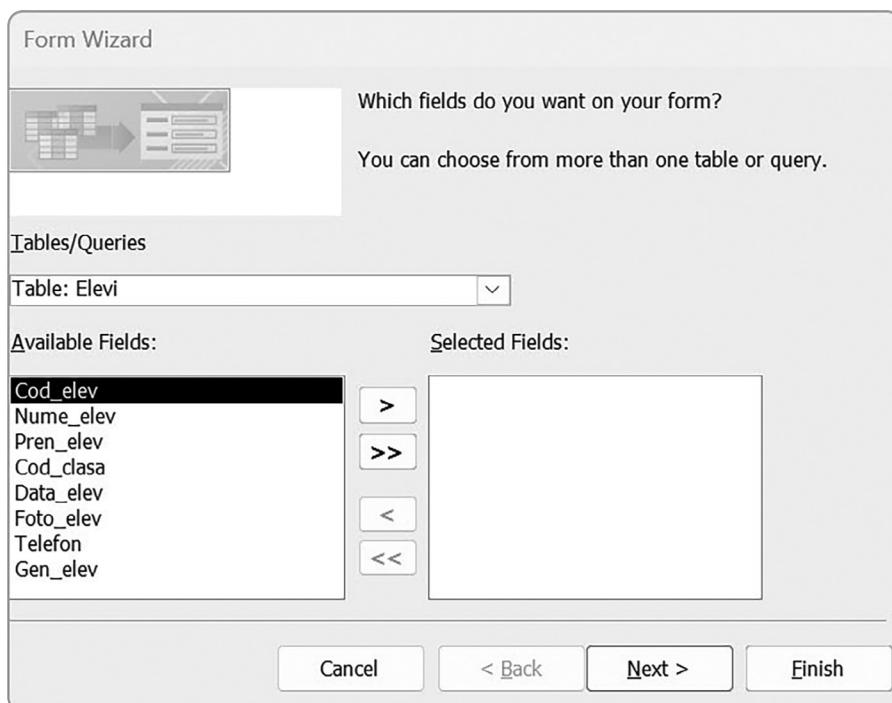


Fig. 4.25. Fereastra de dialog *Form Wizard*  
(Asistentul de formulare)

3. Selectăm câmpurile care vor fi incluse în formular, confirmând alegerea prin apăsarea butonului **>**. Pentru a include toate câmpurile din listă, vom executa un clic pe butonul **>>**. Identificatorii câmpurilor selectate sunt mutați în lista *Selected Fields*. Butoanele **<** și **<<** se folosesc pentru a deplasa înapoi câmpurile greșit alese. Selectăm toate câmpurile și apăsăm butonul *Next*.
4. Următoarea fereastră *Form Wizard* oferă posibilitatea alegerii modului de afișare a câmpurilor în formular (fig. 4.26). Selectăm opțiunea *Columnar* (câmpurile alese la pasul precedent vor fi afișate în formular în coloane).

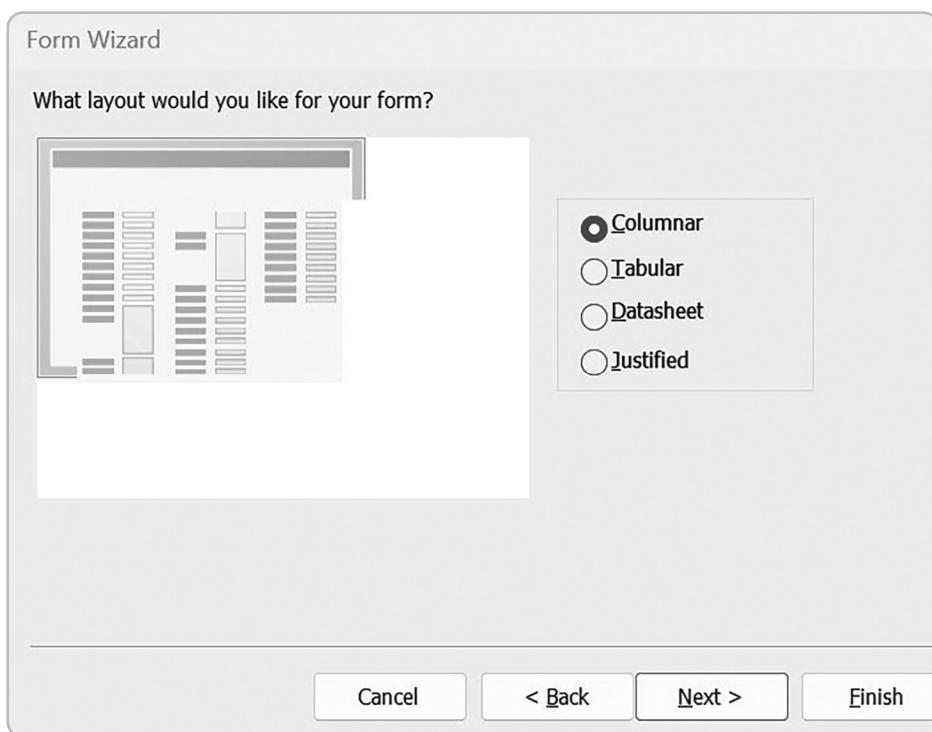


Fig. 4.26. Selectarea modului de afișare a câmpurilor

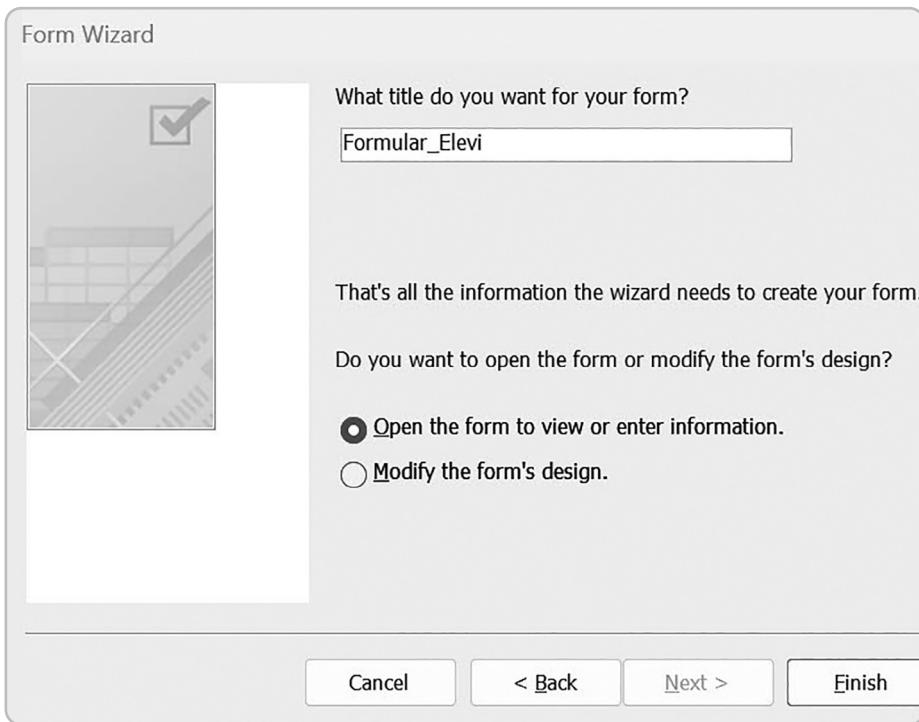


Fig. 4.27. Setarea denumirii formularului

5. În ultima fereastră precizăm numele formularului (*Formular\_Elevi*). Dacă alegem opțiunea *Modify the Form's design* (fig. 4.27), atunci după apăsarea butonului *Finish* formularul va fi deschis în regim de proiectare. În caz contrar, formularul va apărea în regim de editare a datelor.

6. Deschidem formularul pentru a vedea și a introduce informații (fig. 4.28).

The screenshot shows a Microsoft Access form titled "Formular\_Elevi". The form contains the following data:

Cod_elev	e001
Nume_elev	Bacinschi
Pren_elev	Sabina
Cod_clasa	c01
Data_elev	28.09.2007
Foto_elev	
Telefon	022-29-82-54
Gen_elev	F

At the bottom of the form, there is a navigation bar with the text "Record: 1 of 285" and buttons for navigating between records. There are also buttons for "No Filter" and "Search".

Fig. 4.28. Formularul în regim normal

Butoanele din partea de jos a formularului se folosesc pentru a lista înregistrările. Astfel, butoanele , afișează prima și, respectiv, ultima înregistrare. Pentru a vedea precedentă sau următoarea înregistrare față de cea curentă, se va acționa butonul sau, respectiv, .

La inserarea datelor noi se va apăsa butonul sau se va executa *New (blank) Record* din meniul *Insert*.

#### **Observații:**

1. Un formular poate fi creat în baza mai multor obiecte-sursă (tabele și interogări).
2. Modificarea datelor în formular atrage după sine modificarea datelor din obiecte-sursă.

3. Dacă dorim să adăugăm înregistrări noi cu ajutorul unui formular, atunci acesta trebuie să conțină câmpurile cheie primară și câmpurile care au proprietatea *Required* setată Yes. Dacă aceste câmpuri nu vor fi completate, atunci înregistrarea nouă nu va fi acceptată.

## Crearea și modificarea formularelor în regimul de proiectare

Regimul de proiectare este cel mai complet mod de creare sau modificare a formularelor, deoarece în el se poate edita orice element de control al formularului.

1. Pentru a crea în regim de proiectare un formular, executăm un dublu clic pe butonul *Form Design* din meniul *Create*.
2. Apare fereastra de creare (editare) a formularului și meniul *Form Design* cu instrumente pentru editarea formularului (fig. 4.29).
3. În general, suprafața unui formular este formată din câteva secțiuni (fig. 4.29). Redimensionarea fiecărei secțiuni se realizează cu ajutorul mouse-ului.

Formular_Elevi		
Form Header		
Formular_Elevi		
Detail		
1	Cod_elev	Cod_elev
2	Nume_elev	Nume_elev
3	Pren_elev	Pren_elev
4	Cod_clasa	Cod_clasa
5	Data_elev	Data_elev
6	Foto_elev	
7		
8		
9	Telefon	Telefon
10	Gen_elev	Gen_elev

Fig. 4.29. Formularul în regim de proiectare

Linia orizontală (de jos) și linia verticală (la dreapta) stabilesc marginea de jos și, respectiv, marginea dreaptă a formularului.

- Secțiunea principală este **Detail** (apare automat la crearea sau editarea unui formular nou), în care sunt afișate înregistrările din sursa de date a formularului.
- Secțiunile **Form Header** și **Form Footer** reprezintă zona de antet și, respectiv, cea de sub-sol ale formularului, în care, de regulă, se scriu informații explicative sau utile. Aceste informații nu se schimbă de la o înregistrare la alta și pot conține, de exemplu, logouri, sugestii de utilizare a formularului, totaluri, date despre autori, timpul curent, data creării formularului etc.

Secțiunile *Form Header* și *Form Footer* nu apar în mod implicit la editare. Sunt afișate doar dacă în formular se includ elemente din grupul de instrumente *Header/Footer* (al meniului *Form Design*).

4. Adăugarea de câmpuri în formulare se face prin apelarea butonului *Add Existing Fields* (din meniul *Form Design*) după algoritmul menționat anterior (apare fereastra *Field List*).

Grupul de instrumente *Controls* (fig. 4.30) din meniul *Form Design* oferă posibilitatea selectării și plasării elementelor de control pe suprafața formularului.



Fig. 4.30. Grupul de instrumente *Controls*

Să examinăm unele instrumente ale ei:

- Select* nu plasează un oarecare element de control, el se folosește pentru a selecta (atunci când este activ) elemente de control din formular.
- Text Box* creează o casetă care va afișa texte ce vor putea fi modificate de utilizator.
- Label* creează o casetă care poate afișa un text static (ce nu va putea fi modificat de utilizator).
- Button* creează un buton de comandă pentru lansarea la execuție a unei comenzi.
- Tab Control* creează un element de control cu mai multe pagini. Fiecare pagină poate afișa un formular. Cu acest element ușor se realizează comutarea între formulare.
- Link* creează o hiperlegătură.
- Navigation control* creează un subformular de navigare.



*Option Group* creează o casetă dreptunghiulară în care pot fi inserate butoane de opțiune sau casete de validare. În cazul butoanelor, utilizatorul va putea selecta în orice moment de timp un singur buton.



*Insert Page Break* creează un semn pentru imprimantă, determinând-o să treacă la o pagină nouă.



*Combo Box* creează o casetă combinată (se mai spune listă ascunsă) formată dintr-o casetă de text editabilă și o listă derulantă din care se poate alege o valoare.



*Line* creează o linie utilizată pentru design (culoarea și grosimea liniei pot fi modificate).



*Togle Button* creează un buton de comutare pentru activarea sau dezactivarea unei stări.



*List Box* creează o listă derulantă din care se poate alege o valoare.



*Rectangle* creează un dreptunghi utilizat pentru design.



*Check Box* creează o casetă de validare pentru validarea sau interzicerea unei stări.



*Unbound Object Frame* afișează un obiect OLE neasociat, creat cu o aplicație OLE (de exemplu, Microsoft Draw). Nu este o valoare dintr-un câmp OLE a unei înregistrări.



*Attachment* permite inserarea atașamentelor (documente, prezentări, imagini etc.).



*Option Button* creează un buton (se mai spune buton radio) utilizat de asemenea pentru activarea sau dezactivarea unei stări (ori opțiuni). De regulă, se folosește în grup cu mai multe butoane de acest tip pentru organizarea opțiunilor eliminatorii.



*Subform/Subreport* atașează formularului un subformular sau un subraport.



*Bound Object Frame* afișează un obiect OLE – conținut al unui câmp OLE (de exemplu, o imagine).



*Image* afișează o imagine statică (care, fiind plasată în formular, nu va putea fi modificată).



*Web Browser Control* creează o zonă pentru afișarea conținuturilor paginilor web direct în formular.



*Chart* creează o diagramă.

- Din punctul de vedere al funcționalității, deosebim următoarele **categorii ale elementelor de control**:
  - a) *elemente de control legate*;
  - b) *elemente de control calculate*;
  - c) *elemente de control independente*.

**Elementele de control legate** servesc pentru afișarea și editarea datelor câmpurilor tabelelor și interogărilor în baza cărora a fost creat formularul. Un element de control legat este format dintr-un câmp (în care apar datele) și o etichetă asociată acestui câmp (care afișează un

text explicativ, de exemplu, denumirea câmpului). În mod implicit, eticheta asociată afișează identifierul câmpului ale cărui valori se vor afișa de elementul de control (fig. 4.31). În regimul *Datasheet View* în câmpul din figura 4.30 al formularului creat anterior va apărea un număr de telefon.

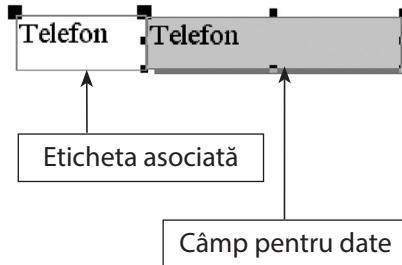


Fig. 4.31. Element de control legat

Componentele elementului de control pot fi redimensionate și deplasate prin metoda *Drag & Drop*.

Adăugarea unui element de control legat se poate face prin accesarea butonului *Add Existing Fields* din meniul *Form Design* sau inserând un element *Text Box* și indicând în caseta *Unbound* numele câmpului necesar.

În pofida denumirii sale, o casetă de text poate afișa, în afară de texte, date numerice, calendaristice etc.

În mod implicit, Access aliniază textul la stânga și numerele la dreapta.

Orice element de control legat poate fi redactat detaliat prin apelarea opțiunii *Properties* din meniul contextual al elementului. La selectarea acestei opțiuni apare fereastra *Property Sheet* cu toate proprietățile elementului selectat din lista derulantă (fig. 4.33).

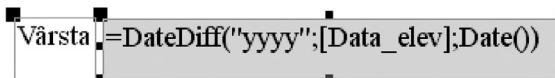


Fig. 4.32. Elementul de control Vârstă elevului

**Elementele de control calculate** se folosesc pentru afișarea valorilor expresiilor și pot fi modificate ca și elementele de control legate. Adăugarea unui astfel de element se face, de asemenea, prin inserarea unui element *Text Box*, doar că în caseta *Unbound* se scrie expresia (precedată de semnul =) a cărei valoare va apărea.

De exemplu, elementul de control din figura 4.32, fiind inserat în formularul creat anterior, va afișa vîrstă elevului.

**Elementele de control independente** nu depind de informațiile din tabele sau interogări, deci ele nu se modifică de la o înregistrare la alta. Se folosesc pentru a crea efecte de design.

- **Proprietățile unui element** de control stabilesc diferite caracteristici ale acestora: culoarea, dimensiunile, sursa de date, poziția în cadrul formularului, comportamentul elementului la unele acțiuni ale utilizatorului etc.

Proprietățile unui element pot fi modificate cu ajutorul ferestrei, care apare la efectuarea unui dublu clic pe acest element (fig. 4.33). Ele sunt grupate în patru categorii: *Format*, *Data*, *Event*, *Other*. Grupul *All* include toate proprietățile, pe care le afișează în ordine alfabetică.

- Grupul **Format** înglobează proprietățile de prezentare a obiectului (dimensiuni, culoare, format etc.).

- Grupul **Data** conține proprietățile referitoare la sursa înregistrărilor care vor fi gestionate de elementul de control.
- Grupul **Event** include lista evenimentelor (acțiuni provocate de utilizator sau de aplicație) la care poate reacționa elementul.
- Grupul **Other** păstrează celelalte proprietăți. Ele se referă la ferestrele Windows.
- Un formular poate fi imprimat ca oricare alt obiect al bazei de date: selectând comanda *Print* din meniul *File*.

Pentru a vedea cum arată formularul înainte de imprimare, se alege comanda *Print/Print Preview* din meniul *File*.



Fig. 4.33. Proprietățile casetei de text Nume\_prof

## Subformulare

Un **subformular** este un formular care se conține în alt formular. Putem include subformulare într-un formular cu ajutorul unui program de asistență sau prin proiectare individuală. Vom examina doar cazul programului de asistență, acesta fiind disponibil pentru apelare doar dacă este activat butonul  *Use Control Wizard* din grupul de instrumente *Controls*.

Fie formularul *Clase* pentru afișarea și modificarea datelor despre clasele bazei de date *Liceu*. Vom adăuga un subformular în care vor apărea informații despre elevii și elevele clasei selectate în formularul principal.

1. Deschidem formularul *Clase* în regimul *Design*. Activăm mai întâi butonul *Control Wizard*, apoi butonul *Subform/Subreport* din grupul de instrumente *Controls* și executăm un clic pe formularul de bază în poziția în care va apărea subformularul. Apare fereastra *SubForm Wizard* din care selectăm tabelul *Elevi*.
2. În următoarea fereastră de dialog alegem câmpurile tabelului *Elevi*, iar în fereastra a treia confirmăm denumirea câmpului de legătură (*Cod\_clasa*) dintre formular și subformular.
3. În ultima fereastră *SubForm Wizard* scriem numele subformularului (*Lista\_elevi*). Activând regimul *Form View*, putem vedea lista elevilor clasei selectate în formularul principal (fig. 4.34).

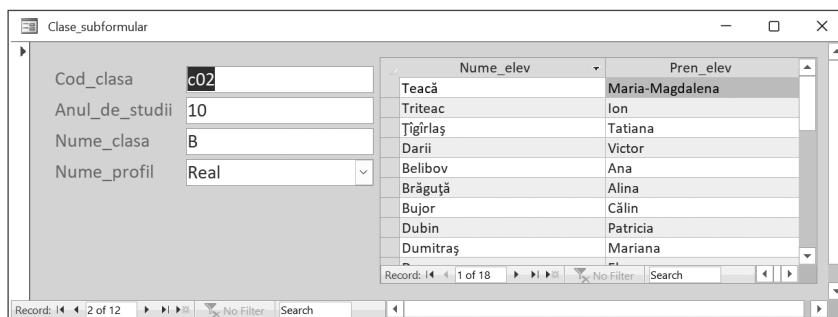


Fig. 4.34. Formular cu subformular

**Observație:** Subformularul creat *Lista\_elevi* automat se va păstra pe disc la salvarea formularului de bază.

## Întrebări și exerciții

- 1 **Explică!** Cu ce scop se utilizează formularurile?
- 2 **Sistematizează-ți cunoștințele!** Ce obiecte ale bazei de date pot fi surse de date pentru formularare?
- 3 **Analizează!** Caracterizați secțiunile formularului în regim de proiectare.
- 4 **Explică!** Care este rolul elementelor de control în cadrul unui formular?
- 5 **Sistematizează-ți cunoștințele!** Descrieți categoriile elementelor de control.
- 6 **Explică!** Care sunt modalitățile de modificare a proprietăților unui element de control?
- 7 **Aplică!** Se consideră baza de date *Liceu*. Creați cu ajutorul unui program de asistență un formular pentru editarea datelor despre:
  - a) profesorii bazei, inclusiv adresa lor;
  - b) disciplinele bazei.

- 8 Exersează!** Adăugați într-un formular creat anterior o casetă de text care va afișa data curentă și timpul curent.
- 9 Exersează!** Adăugați într-un formular creat anterior o casetă de text care va afișa numărul de zile rămase până la sfârșitul: a) anului curent; b) anului școlar curent.
- 10 Învăță să înveți!** Se consideră baza de date *Liceu*. Creați un formular care va afișa datele despre:
- fiecare clasă și va conține un subformular cu lista profesorilor ce predau în această clasă;
  - fiecare clasă și va conține un subformular cu lista disciplinelor studiate în această clasă;
  - fiecare profesor și va conține un subformular cu lista disciplinelor predate de acest profesor.

## 4.14 Rapoarte

**Rapoartele** reprezintă produsul final al unei baze de date. Ele combină date din tabele, interogări și formulare pentru a fi tipărite sau pentru a fi salvate pe disc într-un format atractiv și ușor de citit. Spre deosebire de formulare, rapoartele:

- a) sunt special destinate tipăririi;
  - b) nu pot modifica datele din tabelele sau din interogările care stau la baza lor;
  - c) nu afișează datele în formă tabelară (nu are un regim de tip *Datasheet View*).
- Ca și în cazul formularelор, rapoartele pot fi elaborate cu ajutorul unui program de asistență sau prin proiectare independentă.

### Crearea unui raport cu ajutorul programului de asistență

**1.** În meniul *Create* se află trei butoane pentru crearea rapoartelor:

- *Report* (crearea raportului corespunzător tabelului sau interogării selectate. Este activ doar dacă este selectat un tabel sau o interogare);
- *Report Design* (crearea raportului în regim de proiectare);
- *Blank Report* (crearea unui raport vid, în care urmează a adăuga informație);
- *Report Wizard* (crearea raportului cu ajutorul unui program de asistență).

Executăm un dublu clic pe opțiunea a patra.

**2.** Apare fereastra *Report Wizard* (fig. 4.35). Din lista combinată *Tables/Queries* alegem, pe rând, tabelele *Clase*, *Elevi* și interogarea *Vârsta*, iar din caseta *Available Fields* selectăm câmpurile *Anul\_de\_sudii*, *Nume\_clasa* (ale tabelului *Clase*), *Nume\_elev*, *Pren\_elev* (ale tabelului *Elevi*) și *Vârsta* (al interogării *Varsta*).

**3.** În următoarea fereastră *Report Wizard* selectăm câmpurile (întâi *Anul\_de\_studii*, apoi *Nume\_clasa*), după care se va face gruparea înregistrărilor (fig. 4.36).

**4.** În fereastra a treia *Report Wizard* indicăm câmpurile în conformitate cu care se va face sortarea înregistrărilor (fig. 4.37).

Cu ajutorul butonului *Summary Options* putem specifica pentru fiecare câmp numeric una sau mai multe opțiuni de calcul: suma, media aritmetică, valoarea minimală sau cea maximală. Pentru a calcula vârsta medie a elevilor (pentru fiecare clasă și pe liceu), selectăm opțiunea *Avg* (fig. 4.38). Activarea opțiunii *Summary Only* va afișa doar vârsta medie pe liceu.

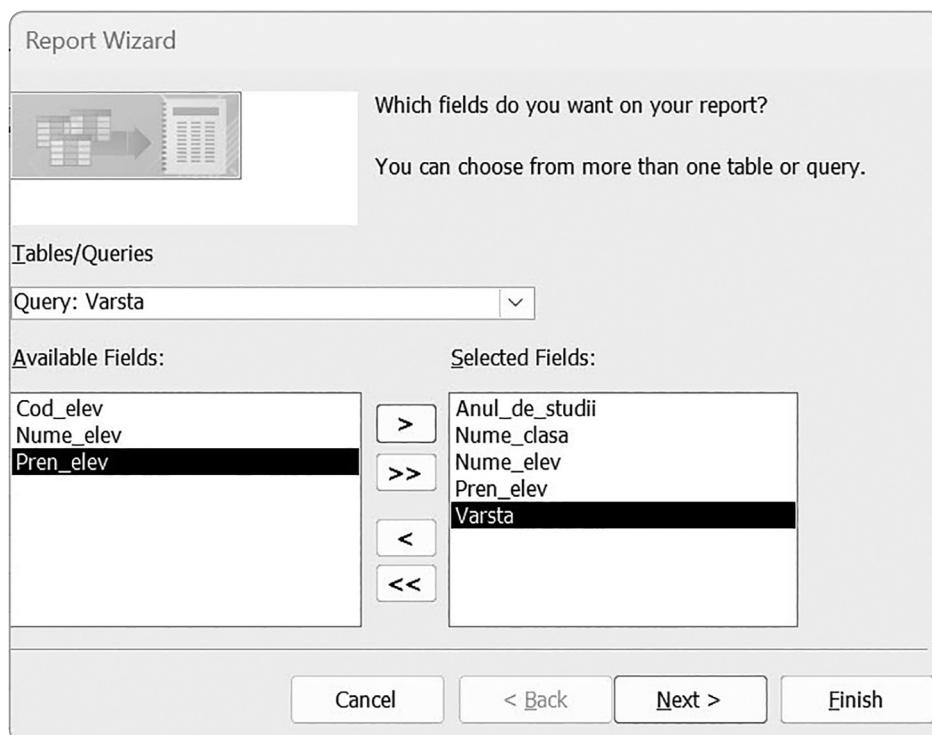


Fig. 4.35. Fereastra de dialog Report Wizard  
(Asistent rapoarte)



Fig. 4.36. Selectarea câmpurilor pentru gruparea înregistrărilor

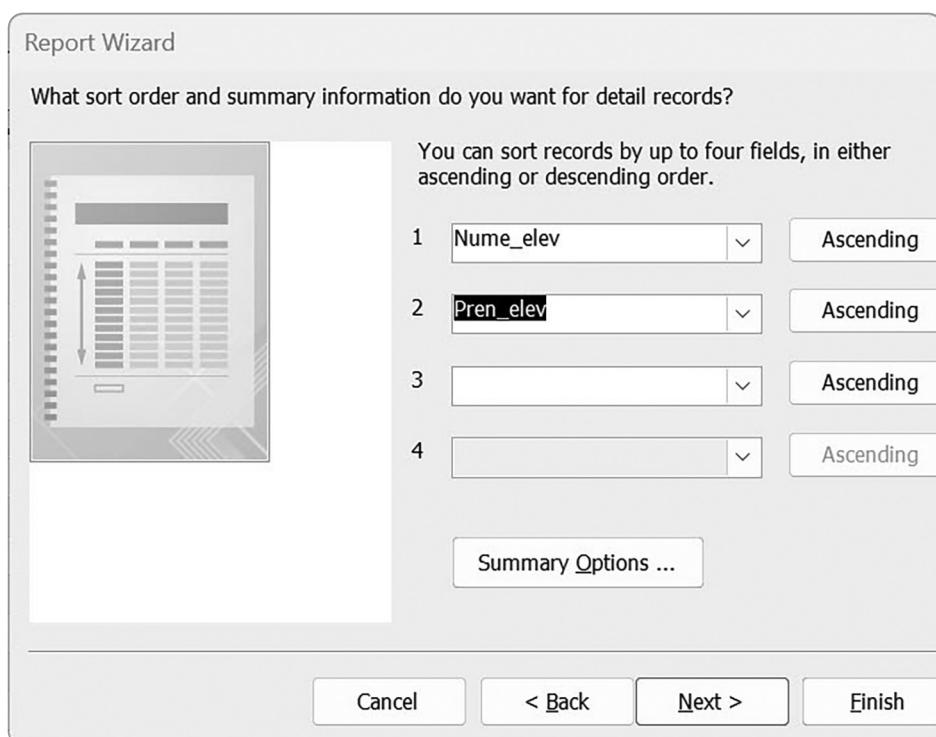


Fig. 4.37 Selectarea câmpurilor pentru sortarea înregistrărilor

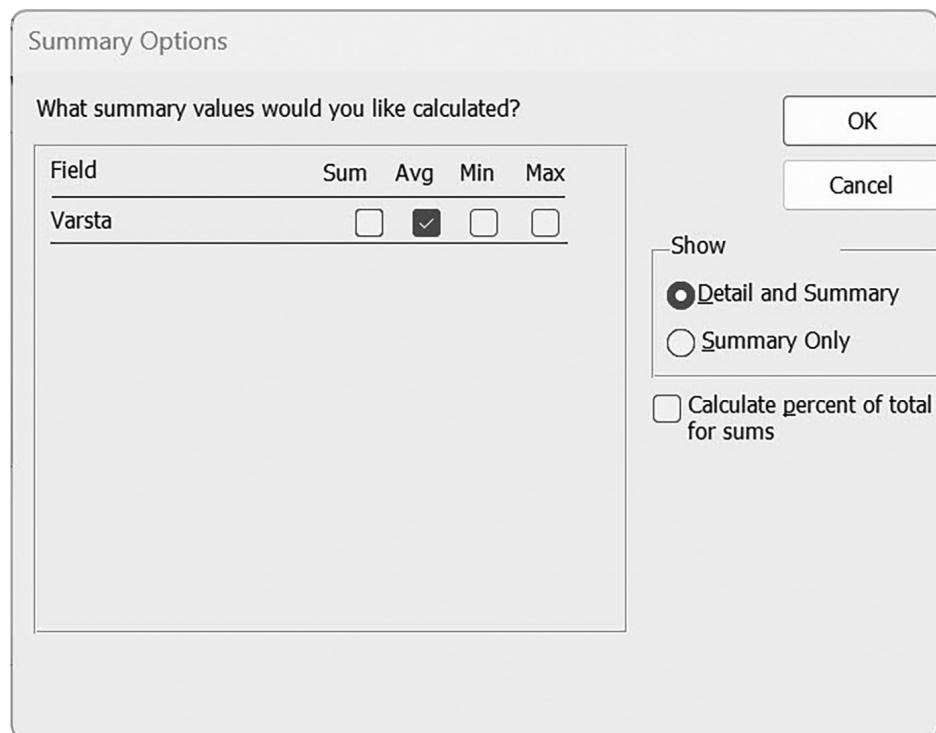


Fig. 4.38. Selectarea opțiunilor de calcul

5. În celelalte trei ferestre Report Wizard specificăm modul de aranjare a înregistrărilor pe pagină, stilul și, respectiv, titlul raportului. În figura 4.39 este reprezentată o parte din raportul creat. Pentru a tipări raportul, se execută un clic pe butonul Print de pe bara de instrumente.
- Observație:** Crearea subrapoartelor se realizează ca și crearea subformularelor.

Anul_de_studii	Nume_clasa	Nume_elev	Pren_elev	Varsta
10	A	Bacinschi	Sabina	17
		Belobrov	Andreea	17
		Brîncă	Carmen	17
		Bulgac	Ion	17

Fig. 4.39. Raportul Vârsta elevilor

### Modificarea rapoartelor în regimul Design View

Regimul *Design View* pentru crearea sau modificarea rapoartelor este asemănător cu regimul *Design View* al formularelor. Ca și formularele, rapoartele conțin 5 secțiuni principale: *Report Header*, *Page Header*, *Detail*, *Page Footer* și *Report Footer*. De asemenea, raportul poate conține și secțiuni pentru grupurile de date. De exemplu, în figura 4.40 este reprezentat raportul creat anterior, deschis în regim de proiectare. Observăm că, în afară de cele 5 secțiuni principale, mai apar secțiunile *Anul\_de\_studii Header*, *Nume\_clasa Header*, *Anul\_de\_studii Footer* și *Nume\_clasa Footer*.

Operațiile cu elementele de control ale rapoartelor se realizează similar operațiilor cu elementele de control din formular.

The screenshot shows the Microsoft Access 'Design View' window for a report named 'Raport\_elevi\_varsta'. The report structure is as follows:

- Report Header:** Contains the title 'Raport\_elevi\_varsta'.
- Page Header:** Contains fields for 'Anul\_de\_studii', 'Nume\_clasa', 'Nume\_elev', 'Pren\_elev', and 'Varsta'.
- Anul\_de\_studii Header:** Contains a field for 'Anul\_de\_studii'.
- Nume\_clasa Header:** Contains a field for 'Nume\_clasa'.
- Detail:** Contains fields for 'Nume\_elev', 'Pren\_elev', and 'Varsta'.
- Nume\_clasa Footer:** Contains a summary formula: `= "Summary for " & "Nume_clasa' = " & " & [Nume_clasa] & "(" & Count(*) & " " & IIf(Count(*)=1,"detail record",""")`. It also contains an 'Avg' control.
- Anul\_de\_studii Footer:** Contains a summary formula: `= "Summary for " & "Anul_de_studii' = " & " & [Anul_de_studii] & "(" & Count(*) & " " & IIf(Count(*)=1,"detail record",""")`. It also contains an 'Avg' control.
- Page Footer:** Contains the formula `=Now()`.
- Report Footer:** Contains the formula `= "Page " & [Page] & " of " & [Pages]`.

Fig. 4.40. Raportul Vârsta elevilor afișat în regimul *Design View* (Regimul de proiectare)

## Crearea diagramelor

Se știe că reprezentarea grafică a datelor este mai ușor înțeleasă și mai ales atunci când se fac comparații între diferite valori numerice.

Într-un sistem de gestiune a bazelor de date diagramele sunt considerate un tip special de rapoarte. Putem crea o diagramă în baza unui tabel sau a unei interogări.

### Exemplu:

Să creăm o diagramă care va reprezenta repartizarea elevilor bazei de date *Liceu* pe clase în baza interogării *Nr\_elevi\_clasa*.

1. Selectăm interogarea *Nr\_elevi\_clasa*. Executăm un clic pe butonul *Form Design* din meniul *Create*. Apare meniul *Form Design*. Ne asigurăm că este activată opțiunea *Use Control Wizard* din grupul de instrumente *Controls*. Selectăm instrumentul *Chart* din același grup de instrumente, apoi executăm un clic în zona formularului în care dorim să apară diagrama.
2. Apare prima fereastră *Chart Wizard* în care selectăm câmpurile *Anul\_de\_studii*, *Nume\_clasa* și *CountOfCod\_elev*.
3. În următoarea fereastră *Chart Wizard* specificăm tipul diagramei: *Column Chart* (diagramă cu coloane).
4. În fereastra a treia *Chart Wizard* (fig. 4.41) stabilim sursa categoriilor de date (*Anul\_de\_studii*), sursa seriilor de date (*CountOfCod\_elev*) și legenda (*Nume\_clasa*).

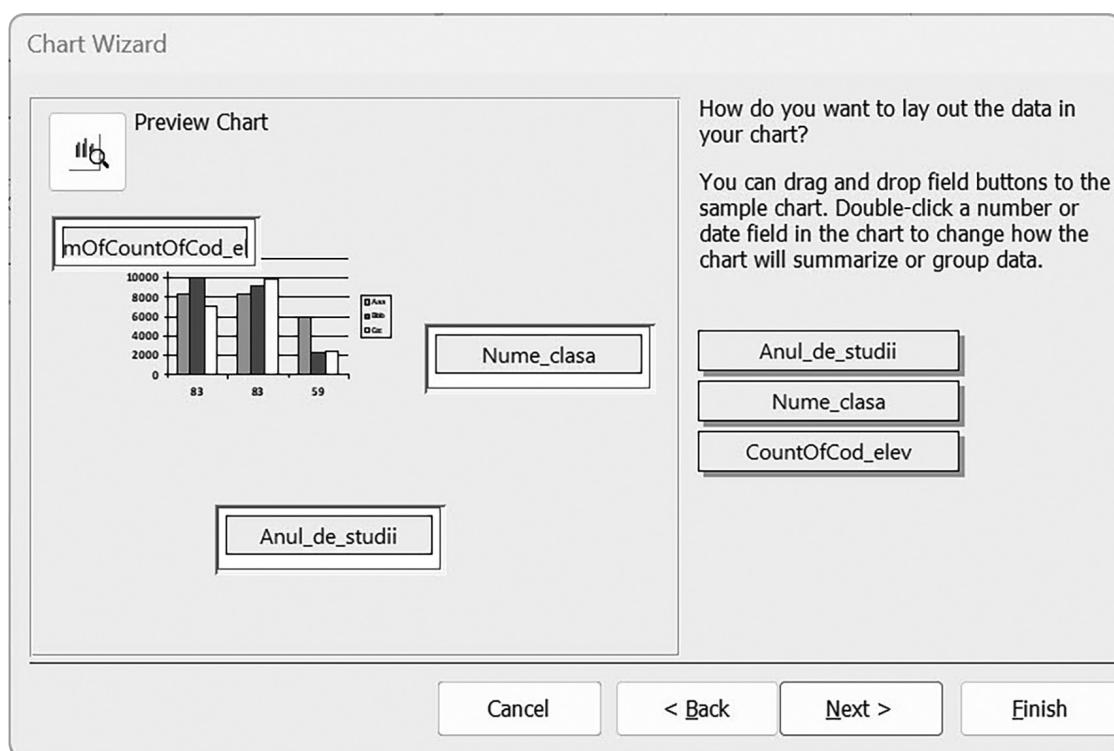


Fig. 4.41. Fereastra de dialog *Chart Wizard* (Asistentul de diagrame)

5. În ultima fereastră *Chart Wizard* atribuim nume diagramei și precizăm dacă va fi sau nu afișată legenda. Obținem diagrama din figura 4.42.

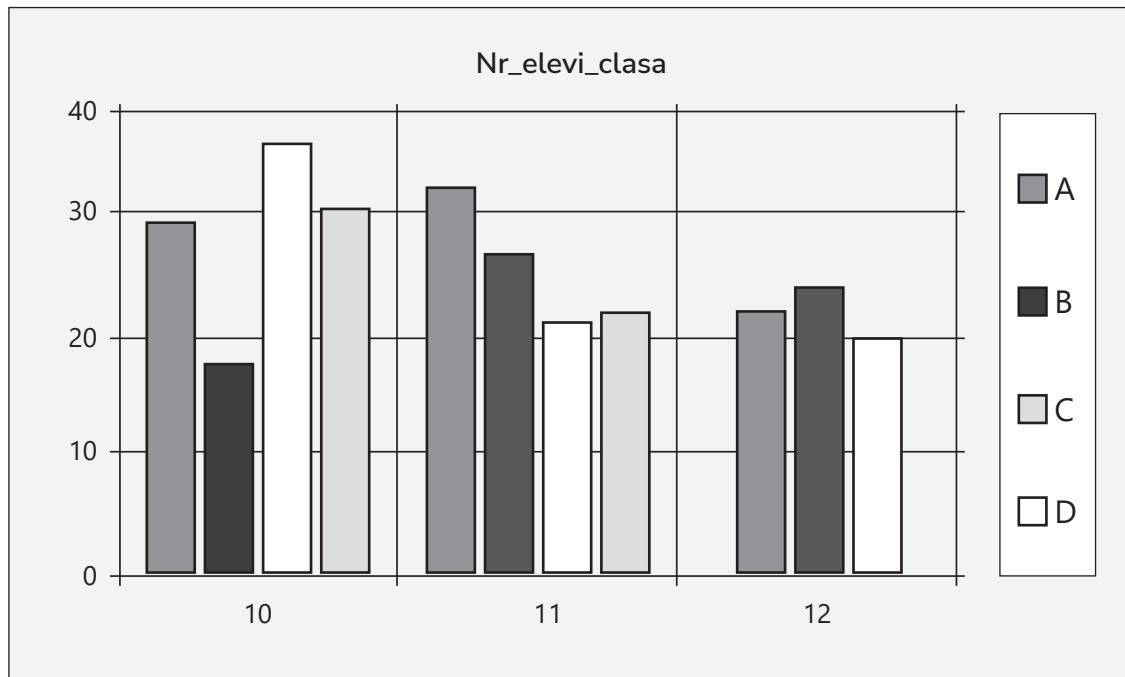


Fig. 4.42. Diagrama care arată repartizarea elevilor bazei de date *Liceu* pe clase

## Întrebări și exerciții

- 1 **Sistematizează-ți cunoștințele!** Cu ce scop se utilizează rapoartele?
- 2 **Explică!** Ce obiecte ale bazei de date pot fi surse de date pentru rapoarte?
- 3 **Analizează!** Care este rolul diagramelor într-o bază de date?
- 4 **Aplică!** Se consideră baza de date *Liceu*. Elaborați un raport în care să fie arătată:
  - a) media aritmetică a orelor realizate săptămânal în fiecare an de studii;
  - b) media aritmetică a orelor realizate săptămânal de fiecare profil;
  - c) suma totală de ore realizate săptămânal în fiecare an de studii și în ansamblu pe liceu.
- 5 **Aplică!** Se consideră baza de date *Liceu*. Creați o diagramă care ar reprezenta grafic numărul:
  - a) de elevi la fiecare profil;
  - b) de elevi din fiecare localitate;
  - c) profesorilor de fiecare gen;
  - d) de ore săptămânal realizat de fiecare clasă.
- 6 **Învață să înveți!** Se consideră baza de date *Liceu*. Creați o diagramă care ar reprezenta grafic numărul de elevi de fiecare vârstă.

## 4.15

## Mentenanța bazelor de date

Pentru ca o bază de date să funcționeze normal, ea trebuie dezvoltată continuu și întreținută. De regulă, aceste operații sunt realizate de administratorul bazei de date. În acest paragraf vom examina unele acțiuni care trebuie efectuate în procesul de exploatare a bazelor de date.

### Compactarea și repararea unei baze de date

- Pe parcursul timpului, în urma adăugării și actualizării datelor, a modificărilor obiectelor bazei, crește volumul fișierului bazei de date și scade viteza de gestionare a bazei.

Aceste consecințe pot fi cauzate nu doar de volumul datelor, dar și de:

- prezența în baza de date a unor obiecte temporare (ascunse de utilizator), chiar dacă Access nu mai are nevoie de ele;
- existența în fișerul bazei de date a unor spații libere, rezultate din ștergerea obiectelor.

În acest caz baza de date are nevoie de *compactare* – operație de înlăturare a fragmentării fișierului bazei –, adică de eliminare a spațiilor și obiectelor neutilizate.

- Uneori fișierul bazei de date se poate deteriora. Acest lucru se poate întâmpla din cauza unor situații extreme (de exemplu, deconectarea sursei de energie, dereglați ale rețelei) sau atunci când mai mulți utilizatori lucrează în mod simultan cu fișierul dat.

În acest caz baza de date are nevoie de *reparație*.

Pentru a **compacta și a repara o bază de date**:

- Închidem baza de date, lăsând deschis sistemul Access.
- Executăm un clic pe instrumentul *Compact and Repair Database* din meniul *Database Tools*.
- Apare caseta de dialog *Database to Compact From*, în care precizăm numele bazei de date ce urmează a fi compactată și reparată. Confirmăm prin apăsarea butonului *Compact*.
- Apare caseta de dialog *Compact Database Into* unde scriem numele sub care se va memora baza de date compactată și reparată. Confirmăm prin apăsarea butonului *Save*.

### Crearea căștilor de rezervă

Access poate repaera complet sau parțial o bază de date deteriorată. În ultimul caz, datele care nu au fost restabileite pot fi recuperate dintr-o copie de rezervă a bazei de date.

Prin urmare, administratorul bazei de date va avea grija să realizeze periodic astfel de căști.

Această acțiune poate fi făcută în mod tradițional (prin copierea fișierului bazei într-un alt dosar).

### Asigurarea securității datelor

O bază de date poate conține informații confidențiale și poate fi destinată mai multor utilizatori, care vor avea acces la ea prin rețea (posibil prin Internet). În această situație se impun măsuri de prevenire a accesului neautorizat. Mai mult chiar, diferiți utilizatori pot avea diferențiate drepturi de acces. De exemplu, unii utilizatori pot avea dreptul să adauge informații, iar alții – doar să vadă unele dintre ele.

În acest scop, unui sau mai mulți utilizatori li se va atribui:

- un nume unic de acces;
- o parolă secretă;
- drepturi de acces sau de proprietar.

Cea mai simplă metodă de protejare a bazei de date de accesul neautorizat este stabilirea unei parole, în lipsa căreia baza de date nu va putea fi deschisă.

Pentru **crearea parolei**:

1. Închidem baza de date, lăsând deschis sistemul Access.
2. Executăm *File* → *Open*. Selectăm fișierul bazei de date, apoi opțiunea *Open Exclusive* din lista derulantă *Open*.
3. Executăm *File* → *Info* → *Encrypt with Password*. Apare caseta de dialog *Set Database Password* în care stabilim și confirmăm parola.

**Observații:**

1. Există și alte acțiuni care pot fi realizate de administratorul bazei de date pentru a asigura protejarea datelor. De exemplu, se poate face **criptarea** bazei, pentru ca datele să nu fie citite cu un editor de texte sau cu utilitare de disc fără a fi decriptate.
2. Crearea grupurilor, parolelor și a drepturilor de acces de asemenea se face apelând submeniul *Security* al meniului *Tools*.

### Creăm, cercetăm și exersăm

În imagine (fig. 4.43) sunt reprezentate relațiile dintre tabelele bazei de date „Club sportiv” (câmpul *Platit* este de tip logic).

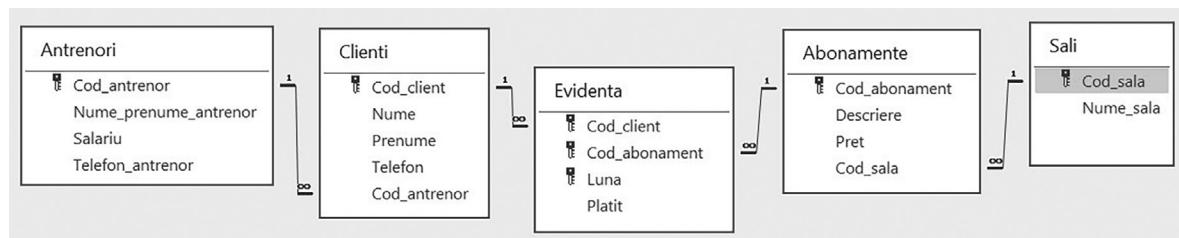


Fig. 4.43. Relațiile dintre tabelele bazei de date „Club sportiv”

1. Creați baza de date conform imaginii și completați-o cu înregistrări.
2. Explicați de ce câmpurile **Cod\_client**, **Cod\_abonament** și **Luna** ale tabelului *Evidenta* formează împreună cheia primară a acestui tabel.
3. Elaborați o interogare care calculează:
  - lista clientilor al căror prenume începe cu litera A sau C;
  - lista antrenorilor al căror nume este format din cel puțin 5 litere;
  - salariul mediu al antrenorilor;
  - lista antrenorilor cu salariul mai mare decât cel mediu;
  - lista codurilor abonamentelor cu preț maxim;
  - numărul de clienti al fiecărui antrenor;
  - numele antrenorilor cu cei mai mulți clienti;
  - suma totală de bani plătită de fiecare client;
  - numele clientilor care au plătit cel mai mult (în total);
  - sălile frecventate cel mai des de clienti.
4. Elaborați formulare de introducere și editare a datelor pentru fiecare dintre tabelele bazei de date „Club sportiv”.

## Creăm în echipă

1. Elaborați un model conceptual relațional al unei baze de date cu informații despre:
  - a) agenda elevului;
  - b) țările Europei;
  - c) automobile;
  - d) titluri de cărți;
  - e) pagini web.
2. Creați tabelele bazei de date proiectate.
3. Stabiliți cheile primare și relațiile dintre tabelele bazei de date.
4. Completăți fiecare tabel cu cel puțin 5 înregistrări.
5. Elaborați:
  - a) 4 interogări de selecție, dintre care una cu parametru;
  - b) 3 interogări de modificare a unor înregistrări;
  - c) 2 interogări de grupare și totalizare;
  - d) o interogare încrucișată;
  - e) o interogare de excludere a unor înregistrări.
6. Creați două formulare pentru introducerea/modificarea înregistrărilor în tabelele bazei de date.
7. Creați cel puțin două rapoarte cu informații din baza de date.
8. Creați o diagramă care va reflecta datele din rapoarte.

# Module la alegere

Pentru a realiza în volum deplin vocația fiecărui dintre voi, curriculumul la Informatică vă oferă posibilitatea să studiați un modul la alegere:

- Prelucrări avansate ale informațiilor din bazele de date.
- Metode experimentale în științele umanistice.
- Programarea Web.
- Structuri dinamice de date în PASCAL sau în C/C++.

Se recomandă ca materialele din modulele respective să fie studiate prin metoda clasei inversate. Amintim că această metodă se bazează pe studiul de sine stătător al elevilor, profesorului revenindu-i rolul de consultant și îndrumător.

Pentru a facilita alegerea de către fiecare dintre elevi a modulului dorit, acest capitol conține o descriere succintă a acestora. Modulele propriu-zise pot fi descărcate de pe pagina oficială a Centrului Tehnologii Informaționale și Comunicaționale în Educație ([www.ctice.gov.md](http://www.ctice.gov.md)), rubrica *Resurse*.

## 5.1 Prelucrări avansate ale informațiilor din bazele de date în formă de liste

### 1. Sortarea și selectarea înregistrărilor

În acest paragraf sunt explicate modalitățile de *sortare și selectare a înregistrărilor* bazelor de date organizate în formă de liste. Astfel de baze de date au fost studiate în clasa a IX-a.

În scopul revocării materiilor studiate și consolidării abilităților de sortare și de selectare a înregistrărilor din astfel de baze de date, recomandăm elevilor să efectueze la calculator lucrările practice prezentate în acest paragraf.

### 2. Rezumarea datelor

Studiind acest paragraf, veți învăța cum pot fi obținute *sintetizări* sau, altfel zis, *rezumări* ale înregistrărilor unei baze de date în formă de liste. Mai exact, veți forma deprinderi de grupare a datelor, de obținere a structurilor imbricate de grupare, a totalurilor și a subtotalurilor etc. pentru informațiile din bazele de date menționate.

În cazul aplicațiilor de gestiune a bazelor de date, rezumarea datelor se efectuează cu ajutorul interogărilor de grupare și ale celor de totalizare, care se folosesc pentru sumarea datelor câmpurilor, pentru obținerea valorilor medii, a valorilor minime sau maxime etc.

### 3. Tabele de sinteză

Foarte des, pentru obținerea totalurilor bazelor de date în formă de liste, sunt folosite *tabele de sinteză*, denumite uneori și *tabele pivot*. Aceste tabele sunt deosebit de utile în cazurile în care listele de date conțin foarte multe, sute și chiar mii de înregistrări.

Tabelele de sinteză sunt similare rezultatelor obținute cu ajutorul interogărilor încrucișate (care pot fi create în aplicații de gestiune a bazelor de date).

Parcurgând acest paragraf, veți lua cunoștință de algoritmul de creare a tabelelor pivot în baza unei liste de date. Menționăm că aceste liste trebuie să corespundă următoarelor cerințe:

- să nu conțină denumiri vide de câmpuri, adică fiecare coloană numai decât trebuie să aibă o denumire;
- să nu aibă rânduri de totalizare.

## 5.2 Metode experimentale în științele umanistice

### 1. Metodologia experimentală a științelor umanistice

Acest paragraf oferă o privire de ansamblu asupra *obiectelor de studiu ale sociologiei*:

- grupurile și colectivitățile umane, de exemplu, populația rurală, populația urbană; grupurile mici (elevii, cadrele didactice, sportivi, ecologii, adolescenții, tinerii, maturii, persoanele în vîrstă și.a.);
- fenomenele și procesele sociale, de exemplu, opinia publică, credințele, tradițiile, educația, politica, comunicațiile;
- instituțiile sociale, cum ar fi familia, școala, parlamentarismul, statul, proprietatea și.a.

În continuare sunt date metodele, tehniciile și procedurile destinate planificării și efectuării cercetărilor sociologice:

- observația;
- experimentul;
- analiza documentelor;
- interviul individual sau de grup;
- sondajul de opinie;
- studiul de caz.

Tot în acest paragraf sunt indicate principalele instrumente informative destinate prelucrării datelor sociologice colectate în cadrul cercetărilor respective.

### 2. Variabile în experimentele sociologice

În acest paragraf sunt explicate variabilele care apar în cadrul unui experiment sociologic, de exemplu:

- motivația elevilor de a învăța;
- starea de bine a elevilor în școală;
- calificarea cadrelor didactice;
- existența laboratoarelor moderne de fizică, chimie, biologie;

- opiniile elevilor referitoare la învățarea anumitor discipline școlare;
- rezultatele învățării demonstate de elevi.

O atenție deosebită se acordă legăturilor probabilistice dintre variabilele ce apar în cadrul unei cercetări sociologice.

**Atenție!** În scopul formării, dezvoltării și consolidării abilităților practice de utilizare a mijloacelor digitale în științele umanistice, recomandăm elevilor ca învățarea să se bazeze pe elaborarea individuală sau în grup a unuia dintre următoarele proiecte:

- 1) Efectuarea unui sondaj în rândul elevilor din cadrul instituției de învățământ privind:
  - a) nivelul de satisfacție de calitatea manualelor;
  - b) nivelul de satisfacție de condițiile fizice din școală;
  - c) gradul de participare a elevilor în activitățile extrașcolare;
  - d) atitudinea elevilor față de eventualele cazuri de copiere;
  - e) activitatea Consiliului elevilor din instituția de învățământ;
  - f) profesiile pentru care ar opta elevii după absolvirea școlii;
  - g) opiniile elevilor privind acțiunile ce ar trebui întreprinse pentru a preveni cazurile de hărțuire (*bullying*), violență școlară, discriminare, sexism, criminalitate juvenilă etc.;
  - h) opiniile elevilor privind acțiunile ce ar trebui întreprinse pentru a promova un mod sănătos de viață, de prevenire a sarcinilor timpurii, de prevenire a bolilor sexual-transmisibile.
- 2) Efectuarea unui sondaj în rândul tinerilor din localitatea în care se află instituția de învățământ privind:
  - a) calitatea serviciilor din localitate destinate tinerilor;
  - b) eventualul loc de construcție a unui centru de agrement;
  - c) eventualul loc de construcție a unui centru sportiv;
  - d) intențiile tinerilor de a contribui finanțiar sau prin muncă neremunerată la amenajarea localității;
  - e) opiniile tinerilor privind acțiunile ce ar trebui întreprinse pentru a preveni cazurile de consum excesiv de alcool, de consum a substanțelor interzise.
- 3) Efectuarea unui sondaj în rândul cetățenilor din localitatea în care se află instituția de învățământ privind:
  - a) calitatea vieții;
  - b) opțiunile politice;
  - c) atitudinea față de anumite evenimente cu rezonanță internațională;
  - d) calitatea drumurilor;
  - e) starea mediului;
  - f) activitatea administrației publice locale etc.

Elaborarea proiectelor va începe în cadrul studierii paragrafului al doilea al modulului în cauză și se va desfășura pas cu pas în procesul învățării fiecărui dintre paragrafele următoare.

### 3. Grupurile de control și variabilele-parazit

Studiind acest paragraf, veți afla semnificația principalilor termeni utilizați în cercetările sociologice:

- grup de control;
- experiență sincronică;
- experiență diacronică;
- grup de control cu artefact;
- variabilă-parazit.

De asemenea, vă veți forma și dezvolta abilitățile de identificare a grupurilor de control și a tipului de experiment în funcție de specificul fenomenului social studiat.

### 4. Planurile experimentale și alegerea subiecților

Studiind acest paragraf, vă veți familiariza cu tipurile de planuri experimentale – unifactoriale și multifactoriale – și vă veți dezvolta abilitățile de alegere a subiecților în funcție de specificul fenomenului social supus cercetării.

### 5. Descrierea matematică a informațiilor primare

Materiile din acest paragraf constituie baza matematică a cercetărilor sociologice cantitative și asigură o conexiune între științele umanistice și cele reale, oferind sociologului instrumente relevante, obiective și veridice de măsurare a intensității fenomenelor sociale. După studierea acestui paragraf, veți putea face o alegere argumentată a tipului scalei de măsură (nominală, ordinală, de intervale, de raport) în funcție de specificul fenomenului social studiat și vă veți dezvolta competența de proiectare a scalelor respective.

### 6. Rezumatul și descrierea numerică a datelor

Scopul acestui paragraf constă în formarea și dezvoltarea competenței de rezumare și descriere numerică a datelor utilizate în cadrul cercetărilor sociologice cantitative. Este important să înțelegeți cum utilizarea datelor numerice contribuie la fundamentarea pe dovezi a concluziilor referitoare la un anumit fenomen social.

În acest paragraf, pentru fiecare tip de scală, sunt introduse în studiu indicii de tendință centrală și indicii de dispersie și pe baza a mai multor exemple se ilustrează modul de interpretare a valorilor numerice ale acestora.

### 7. Populații și eșantioane

După studierea acestui paragraf veți stăpâni metodele de prelevare a eșantioanelor:

- empirice;
- de cote;
- de unități-tip;
- probabilistice;
- tragere la sorți;
- de stratificare.

## **8. Analiza datelor cu ajutorul aplicațiilor de calcul tabelar**

Scopul acestui paragraf constă în formarea și dezvoltarea abilităților de utilizare a facilităților avansate ale aplicațiilor de calcul tabelar pentru analiza datelor din domeniul științelor umanistice:

- colectarea datelor;
- verificarea datelor;
- sistematizarea datelor,
- gruparea datelor;
- calcularea indicilor de tendință centrală;
- crearea de histograme;
- interpretarea rezultatelor obținute.

## **9. Produse-program pentru științele sociale**

De obicei, aplicațiile specializate destinate planificării și desfășurării cercetărilor sociologice și, respectiv, prelucrării datelor colectate sunt produse comerciale. În consecință, utilizarea lor presupune achitarea unui costuri fie pentru instalarea lor pe calculatoarele școlii sau ale elevilor, fie pentru procurarea abonamentelor de utilizare a acestora prin Internet. Totodată, există și produse-program libere (fără plată), însă instrumentele oferite de ele sunt mai modeste. Prin urmare, elevii, sub îndrumarea cadrului didactic, vor alege produsul-program ce va fi utilizat în procesul de predare-învățare-evaluare și, cu ajutorul sistemului de asistență, vor studia instrumentarul respectiv.

**Atenție!** Proiectele elaborate de către elevi vor fi prezentate în cadrul unor sesiuni de evaluare reciprocă. Aceste sesiuni vor include prezentări electronice cu capturi de ecran ce confirmă corectitudinea cercetărilor sociologice, concluziile ce derivă din aceste cercetări, răspunsuri la întrebările și comentariile formulate de către cei prezenți la sesiunile de evaluare reciprocă.

## **5.3 Programarea Web**

### **1. Modalități de utilizare a codurilor JavaScript în documente HTML**

Acest paragraf include informațiile de bază referitoare la limbajul de programare JavaScript (utilizat în dezvoltarea diferitelor tipuri de aplicații web interactive) și la modalitățile de utilizare a codurilor JavaScript în documente hipertext.

Sunt examineate trei cazuri:

- inserarea codului JavaScript direct în documentul HTML;
- importarea codurilor JavaScript dintr-un fișier separat (care conține doar cod JavaScript);
- utilizarea codurilor JavaScript la precizarea atributelor etichetelor HTML.

### **2. Date simple, expresii și funcții**

În acest paragraf este explicată sintaxa de bază a expresiilor JavaScript.

O expresie JavaScript este o combinație validă de operatori și operanzi. Sunt studiate următoarele cazuri particulare de expresii:

- a) constantele (numerice, inclusiv constantele obiectului Math; şiruri de caractere; logice; constanta *null*);
- b) variabilele;
- c) masivele şi elementele lor;
- d) apelurile de funcţii (a celor de tip *return*, inclusiv funcţiile matematice – metode ale obiectului Math);
- e) expresiile aritmetice;
- f) expresiile logice;
- g) combinaţiile corecte ale expresiilor a)–f) cu ajutorul operatorilor şi al parantezelor.

În continuare, tot în acest paragraf sunt prezentate diferite tipuri de operatori JavaScript:

- aritmetici;
- de atribuire;
- de incrementare;
- de decrementare;
- de comparare;
- logici.

### **3. Operaţii de intrare-iesire**

În acest paragraf veţi studia câteva modalităţi de organizare a procesului de introducere şi de citire a datelor (sau extragere a rezultatelor) pentru aplicaţii web:

- cu metoda *write* a obiectului *document*;
- prin intermediul casetelor de text;
- cu ajutorul ferestrelor de dialog.

### **4. Structuri de date: masive; şiruri de caractere**

Studiind acest paragraf, veţi afla că în JavaScript masivele, deci şi şirurile de caractere sunt *obiecte* – structuri formate din *proprietăţi* (datele obiectului) şi *metode* (funcţiile ale obiectului).

Veţi studia metodele de prelucrare a masivelor (concatenarea masivelor, eliminarea elementelor, adăugarea elementelor, modificarea poziţiilor elementelor, ordonarea elementelor, extragerea unei secvenţe de masiv etc.) şi operaţiile cu şiruri de caractere (determinarea lungimii şirului, concatenarea şirurilor, apelarea simbolului după indicele lui, aflarea codului ASCII al unui simbol, aflarea simbolurilor după codul lor, aflarea poziţiei subşirului în şir, extragerea unui subşir din şir, compararea şirurilor, separarea subşirurilor, convertirea literelor şirului, formatarea şirurilor, transformarea şirului în număr etc.).

### **5. Controlul acţiunilor. Evenimente**

În acest paragraf veţi lua cunoştinţă cu noţiunea de *eveniment* (în contextul programării). Evenimentele se utilizează pentru a oferi posibilitatea utilizatorului să interacţioneze cu aplicaţiile web: apăsarea unei taste, a unui buton de comandă sau a unităţii de mouse, mişcarea mouse-ului, plasarea cursorului asupra unui element al paginii web, selectarea unui câmp de formular, accesarea unei hiperlegături etc.

## **6. Structuri de control**

Structurile de control reprezintă o caracteristică de bază a limbajelor de programare. În acest paragraf veți învăța cum, utilizând sintaxa JavaScript, pot fi create următoarele structuri de control:

- expresie condiționată;
- structuri decizionale cu operatorul *if ... else*;
- selectorul;
- ciclu cu precondiție;
- ciclu cu postcondiție;
- ciclu cu parametru;
- ciclul *for... in*;
- ciclu cu parametri (ciclul *for*).

De asemenea, toate paragrafele descrise mai sus conțin:

- întrebări pentru autoevaluare (menite să contribuie la conștientizarea și consolidarea celor studiate);
- diverse exerciții de integrare și de aplicare a cunoștințelor noi;
- sarcini mai complexe, care au rolul de a vă forma deprinderi de bază pentru scrierea codurilor-program JavaScript și de elaborare a aplicației web pentru soluționarea unor probleme reale, frecvent întâlnite în activitatea cotidiană.

## **5.4 Structuri dinamice de date în PASCAL**

### **1. Variabile dinamice. Tipul referință**

Acest paragraf include informațiile de bază referitoare la clasificarea variabilelor unui program de calculator în variabile statice și variabile dinamice. Este explicitat modul de referire a variabilelor dinamice și tipurile speciale de date, utilizate în acest scop.

La studierea acestui paragraf este important să înțelegeți diferența dintre modurile de alocare și de eliberare a memoriei în cazul variabilelor statice și al variabilelor dinamice, să vă formați și să vă dezvoltați abilitățile de creare și de distrugere a variabilelor dinamice.

### **2. Structuri de date**

Studiind acest paragraf, veți afla cum se clasifică structurile de date utilizate în programele de calculator:

- structuri implicate și structuri explicite;
- structuri statice și structuri dinamice;
- structuri omogene și structuri eterogene;
- structuri recursive.

Paragraful reprezintă o scurtă introducere în structurile de date și are drept scop înțelegerea rolului *informației de structură* în descrierea relațiilor ce există între componentele datelor structurate.

### 3. Liste unidirectionale

Studiind acest paragraf, vă veți familiariza cu părțile componente ale structurilor dinamice de date: câmpul datelor și câmpul legăturilor, veți învăța cum să construiți listele unidirectionale prin metoda iteratăie și prin metoda recursiei.

### 4. Prelucrarea listelor unidirectionale

Studierea acestui paragraf va necesita din partea elevilor o atenție sporită, întrucât operațiile frecvent utilizate în cazul listelor unidirectionale se regăsesc și în prelucrarea celorlalte structuri dinamice de date:

- parcurgerea listei și prelucrarea informației utile, asociate fiecărei celule;
- căutarea unui anumit element, identificat prin valoarea sa;
- includerea (inserarea) unui element într-un anumit loc din listă;
- excluderea (ștergerea) unui element dintr-o listă și.a.

Reprezentările grafice ale listelor unidirectionale și ale operațiilor efectuate asupra acestora, incluse în paragraful respectiv, vă vor ajuta să înțelegeți într-un mod intuitiv esența operațiilor de prelucrare a datelor stocate în structurile dinamice de date și modificarea structurilor propriu-zise.

### 5. Stiva

Prin stivă (în limba engleză *stack*) înțelegem o listă unidirectională cu proprietatea că operațiile de introducere și extragere a elementelor se fac la un singur capăt al ei. Deși stiva reprezintă un caz particular al listelor unidirectionale, ele au o importanță practică deosebită, fiind utilizate atât la elaborarea programelor de sistem (gestionarea memoriei, implementarea apelurilor de subprograme, cursia și.a.), cât și la elaborarea programelor de aplicații (analiza sintactică, simularea proceselor economice, gestionarea documentelor electronice etc.).

La studierea acestui paragraf elevii se vor concentra asupra utilizării stivelor pentru rezolvarea problemelor frecvent întâlnite în activitatea cotidiană a economiștilor, inginerilor, programatorilor.

### 6. Cozi

Prin coadă (în engleză *queue*) înțelegem o listă unidirectională în care toate introducerile se efectuează la unul dintre capete, iar extragerile se efectuează la celălalt capăt. Ca și stivele, cozile reprezintă un caz particular al listelor unidirectionale și sunt utilizate, în special, pentru simularea proceselor de deservire a clientilor, de gestionare a circulației mijloacelor de transport, de planificare a lucrărilor.

### 7. Arbori binari

Comparativ cu listele unidirectionale, arborii binari sunt structuri mai complexe, iar studierea lor necesită stăpânirea în volum deplin atât a tehnicilor iterative, cât și a celor recursive. Prin urmare, înainte de studierea acestui paragraf, elevii își vor împrospăta cunoștințele respective, punând accentul pe specificul definițiilor recursive și modul de gestionare a memoriei în cazul apelurilor recursive.

Reprezentările grafice incluse în acest paragraf vor ajuta elevii să înțeleagă mai bine modul de organizare a datelor în formă de arbori binari și semnificația următorilor termeni:

- nod, nod rădăcină, nod terminal, nod neterminat;
- descendant, descendantul stâng, descendantul drept;
- subarbore, subarborele stâng, subarborele drept;
- nivelul nodului, înălțimea arborelui.

## **8. Parcurgerea arborilor binari**

Parcurgerea arborilor binari presupune vizitarea nodurilor acestora într-o anumită ordine. După studierea acestui paragraf veți putea elabora subprograme de parcurgere a arborilor binari în:

- preordine (rădăcina → subarborele stâng → subarborele drept);
- inordine (subarborele stâng → rădăcina → subarborele drept);
- postordine (subarborele stâng → subarborele drept → rădăcina).

Algoritmii de parcurgere a arborilor binari au o largă utilizare practică, în special, la evaluarea expresiilor scrise în limbajele de programare de nivel înalt.

### **Activități recomandate de învățare**

În scopul formării și dezvoltării abilităților practice de utilizare a structurilor dinamice de date, recomandăm elevilor următoarele activități de învățare:

*Exerciții de:*

- introducere intuitivă (prin desen) a metodelor de alocare dinamică a memoriei;
- argumentare a necesităților de utilizare a structurilor dinamice de date;
- evidențierea diferențelor dintre structurile implicate și structurile explicite de date, dintre structurile omogene și structurile eterogene de date, dintre structurile statice și structurile dinamice de date;
- selecție a problemelor, soluționarea cărora necesită utilizarea structurilor de date propuse în studiu;
- creare, utilizare și distrugere a variabilelor dinamice;
- elaborare a programelor în care se utilizează variabile dinamice;
- explicare a modului de alocare a memoriei operative în cazul utilizării variabilelor statice și a variabilelor dinamice;
- stocare și de prelucrare a datelor cu ajutorul listelor, stivelor, cozilor și arborilor binari.

*Studii de caz:*

- căutarea informației în liste, cozi, stive și arborii binari;
- parcurgerea listelor, stivelor, cozilor și a arborilor binari;
- inserarea și eliminarea datelor din liste, stive, cozi și arborii binari;
- domeniile de utilizare a structurilor dinamice de date.

*Proiecte:*

- prelucrarea listelor de candidați în cazul admiterii la liceu;
- prelucrarea listelor de cuvinte distințe ce se întâlnesc într-un text;
- vizualizarea fluxului de intrare- ieșire a vagoanelor în cazul unui depou feroviar;

- prelucrarea listelor de angajați ai unei întreprinderi;
- analiza sintactică a expresiilor aritmetice;
- vizualizarea firului de așteptare în cazul avioanelor ce solicită aterizarea într-un aeroport;
- crearea și prelucrarea arborilor binari ce intervin în cazul turneelor sportive „prin eliminare”;
- evaluarea expresiilor aritmetice, reprezentate prin arbori binari.

Programele, studiile și proiectele elaborate de către elevi vor fi prezentate în cadrul unor sesiuni de evaluare reciprocă. Aceste sesiuni vor include prezentări electronice cu capturi de ecran ce confirmă corectitudinea sintactică și logică a programelor elaborate, rezultatele afișate de aceste programe, demonstrări practice pe calculator.

## 5.5 Structuri dinamice de date în C/C++

### 1. Pointeri și referințe

De obicei, accesarea valorilor unei variabile se face prin indicarea numelui ei. Însă, în programele de o reală importanță practică, apare necesitatea utilizării nu doar a valorilor curente ale variabilelor declarate în program, dar și a adreselor locațiilor de memorie, alocate acestor variabile respective.

În scopul prelucrării adreselor de memorie, limbajele C/C++ oferă programatorului tipuri speciale de date, denumite *pointeri*. Multimea de valori ale unei variabile de tip *pointer* este formată din adrese de memorie. Amintim că în limba engleză cuvântul *pointer* semnifică „indicator, arătător”;

Utilizarea variabilelor de tip *pointer* este deosebit de utilă în cazurile în care programatorul dorește să prelucreze în programele C/C++ nu doar valorile curente ale anumitor variabile, dar și legăturile ce există între variabilele respective.

### 2. Structuri de date

În programele C/C++ structurile de date conțin nu doar valorile anumitor variabile, dar și relațiile (legăturile) ce există între ele. Cea mai simplă structură de date este tabloul format din componente de același tip. Legăturile dintre componentele unui tablou sunt redată prin ordonarea și numerotarea acestora (pe rânduri și pe coloane în cazul tablourilor bidimensionale).

În cazul unor structuri de date mai complexe, componentele respective pot fi de cele mai diverse tipuri, iar legăturile dintre ele se indică cu ajutorul pointerilor. Mai mult ca atât, componentele unei astfel structuri de date pot fi create și eliminate în procesul execuției programelor în care ele au fost declarate. Structurile de date componente cărora conțin și pointeri se numesc *structuri dinamice de date*.

### 3. Liste unidirecționale

Fiecare componentă a acestei structuri de date conține două câmpuri: câmpul datelor de prelucrat și câmpul adreselor. Evident, câmpul adreselor este de tip *pointer*. Pointerul respectiv indică următoarea componentă din listă, iar în cazul ultimei componente a listei are valoarea NULL.

Studiind acest paragraf, vă veți forma și dezvolta abilitățile de prelucrare a listelor unidirectionale:

- inițializarea listei;
- parcurgerea listei;
- includerea componentelor în listă;
- eliminarea componentelor din listă.

#### **4. Coada și stiva alocate dinamic**

Cozile și stivele reprezintă cazuri particulare ale listelor. Spre deosebire de listele convenționale, pentru care inserarea și eliminarea componentelor poate fi făcută în orice loc al listei, în cazul cozilor și al stivelor, operațiile respective se fac doar la extremități: la începutul și/sau la sfârșitul listei.

În cazul cozilor, noile componente se inserează la o extremitate a listei, iar eliminarea – la extremitatea opusă. Din aceste considerente, cozile mai sunt denumite structuri de date de tipul *First In – First Out* (Primul venit – primul deservit).

În cazul stivelor, noile componente se inserează și se elimină doar la una dintre extremități. Din aceste considerente, stivele mai sunt denumite structuri de date de tipul *Last In – First Out* (Ultimul venit – primul deservit).

#### **5. Arbo里 binari (de căutare)**

Componentele arborilor binari sunt formate din trei câmpuri:

- câmpul datelor de prelucrat;
- câmpul ce conține adresa subarborelui stâng;
- câmpul ce conține adresa subarborelui drept.

În cazul în care unul dintre câmpurile de adrese conține valoarea NULL, subarborele respectiv lipsește. În cazul în care ambele câmpuri de adrese conțin valorile NULL, componenta respectivă este una terminală, având denumirea alegorică „frunză”.

Studiind acest paragraf, vă veți forma și dezvolta abilitățile de prelucrare a arborilor binari:

- inițializarea arborilor;
- inserarea și eliminarea componentelor;
- parcurgerea arborilor în preordine, inordine și postordine.

#### **Activități recomandate de învățare**

În scopul formării și dezvoltării abilităților practice de utilizare a structurilor dinamice de date, recomandăm elevilor activități de învățare listate la sfârșitul paragrafului precedent.

# Extensi

Materiile din acest capitol sunt destinate studierii aprofundate a următoarelor teme din Informatică:

<b>Tehnici de programare:</b>	Metoda reluării. Metoda desparte și stăpânește.
<b>Modelare și calcul numeric:</b>	Metoda lui Newton. Metoda trapezelor.
<b>Structuri dinamice de date:</b>	Tipul de date pointer (PASCAL).

Temele respective, prevăzute de curriculum, au statut opțional și sunt propuse elevilor care, după absolvirea liceului, intenționează să-și continue studiile în domeniul informaticii și tehnologiei informației.

Ca și în cazul modulelor la alegere, se recomandă ca temele din acest capitol să fie studiate prin metoda clasei inverse. Amintim că această metodă se bazează pe studiul de sine stătător al elevilor, profesorului revenindu-i rolul de consultant și îndrumător.

Pentru a facilita alegerea de către fiecare dintr-elevi a temelor dorite, mai jos este prezentată o descriere succintă a acestora. Temele propriu-zise pot fi descărcate de pe pagina oficială a Centrului Tehnologii Informaționale și Comunicaționale în Educație ([www.ctice.gov.md](http://www.ctice.gov.md)), rubrica *Resurse*.

## 1. Metoda reluării

Ideea acestei tehnici de programare (în engleză *backtracking* – darea înapoi) este foarte simplă:

- soluția problemei este reprezentată printr-un vector (un tablou);
- componentele vectorului se aleg pas cu pas din multimi ordonate, câte o multime pentru fiecare dintre componente;
- un element din multimea curentă este inclus ca o componentă în vectorul în curs de construcție doar atunci când el satisfac anumite condiții de continuare;
- dacă în multimea curentă astfel de elemente nu mai există, se revine la alegerea precedentă.

Mentionăm faptul că anume revenirea la alegerea precedentă a și dat denumirea metodei studiate.

Deși nu garantează obținerea unei soluții optime, metoda reluării este pe larg folosită pentru elaborarea programelor destinate prelucrărilor grafice, trasării rutelor, distribuirii resurselor și.a.

## **2. Metoda Desparte și stăpânește**

Metoda *Desparte și stăpânește* (în latină *Divide et impera*) este o metodă generală de elaborare a algoritmilor, care presupune:

- împărțirea repetată a unei probleme de dimensiuni mari în două sau mai multe subprobleme de același tip, dar de dimensiuni mai mici;
- rezolvarea subproblemelor în mod direct, dacă dimensiunea lor permite aceasta, sau împărțirea lor în alte subprobleme de dimensiuni și mai mici;
- combinarea soluțiilor subproblemelor rezolvate pentru a obține soluția problemei inițiale.

De obicei, această metodă este folosită foarte des pentru căutarea binară, sortarea elementelor, estimarea complexității și planificarea lucrărilor.

Programele elaborate în baza metodei *Desparte și stăpânește* sunt simple, iar timpul de execuție este relativ mic.

## **3. Metoda Newton**

Metoda Newton, la fel ca și metoda coardelor, se aplică pentru rezolvarea ecuațiilor algebrice și transcendentale. Această metodă este mai rapidă decât metoda coardelor, însă ea poate fi aplicată doar pentru anumite ecuații de tipul  $f(x) = 0$ .

Astfel, pe intervalul  $[a, b]$ , pe care se caută soluția respectivă, funcția  $f(x)$  trebuie să îndeplinească următoarele condiții:

- să fie continuă;
- valorile  $f(a)$  și  $f(b)$  să aibă semne diferite;
- are prima și a doua derivată, ambele continue, cu semn constant.

Formula de recurență pentru calcularea aproximărilor soluției conține atât funcția  $f(x)$ , cât și prima derivată a acesteia.

Intuitiv, metoda în cauză constă în trasarea consecutivă a tangentelor la graficul funcției  $f(x)$  și identificarea punctelor de intersecție ale tangentelor respective cu axa  $0x$ .

Trasarea iterativă a tangentelor se repetă până când distanța dintre două puncte succesive de intersecție devine mai mică decât precizia cu care trebuie calculată soluția ecuației.

## **4. Metoda trapezelor**

Metoda trapezelor este o tehnică de integrare numerică folosită pentru a approxima valoarea unei integrale definite. Metoda se bazează pe împărțirea intervalului de integrare în subintervale mai mici, de obicei de lungime egală, și approximarea ariei sub graficul funcției  $f(x)$  prin calcularea ariilor trapezelor înscrise sub graficul acesteia.

Intuitiv, metoda trapezelor constă în următoarele:

- intervalul de integrare  $[a, b]$  este împărțit în subintervale egale;
- pentru fiecare subinterval se construiește un trapez, format din axa  $0x$ , liniile perpendiculare pe axa  $0x$ , ce trec prin extremitățile subintervalului respectiv, și approximarea liniară a funcției  $f(x)$  pe acest subinterval;
- însumarea ariilor trapezelor construite.

Întrucât approximarea prin trapeze este mai precisă decât approximarea prin dreptunghiuri, metoda trapezelor necesită mai puține calcule decât cea a dreptunghiurilor. În consecință, pentru una și aceeași precizie de calculare a valorilor approximative ale integralelor definite, metoda trapezelor este mai rapidă.

## 5. Tipul de date pointer

Această temă va ajuta elevii să înțeleagă cum programatorii pot gestiona în mod eficient memoria calculatorului, având controlul complet asupra creării, distrugerii și prelucrării variabilelor dinamice, a alocării și eliberării memoriei operative utilizate de astfel de variabile.

Tipul de date pointer, uneori sub alte denumiri, se întâlnește practic în toate limbajele moderne de programare, fiind un instrument frecvent utilizat în elaborarea programelor destinate prelucrării structurilor complexe de date.

Formându-și, dezvoltându-și și consolidându-și abilitățile de a opera cu pointeri, elevii își vor crea o bază solidă pentru studierea limbajelor moderne de programare, destinate elaborării produselor-program de sistem, a aplicațiilor multimedia, a jocurilor de calculator.



ANATOL GREMALSCHI  
SERGIU CORLAT  
ANDREI BRAICOV

XII

# Informatică

Manual pentru clasa a XII-a

