BAZE DE NUMERAȚIE

prof. dr. Daniela Marcu
prof. Ovidiu Marian Marcu

Introducere

1. Ce este un sistem de numerație?

Un sistem de numerație reprezintă o metodă de scriere și reprezentare a numerelor folosind un anumit set de simboluri și reguli de formare a acestora.

În viața de zi cu zi utilizăm cel mai frecvent sistemul de numerație zecimal, care are baza 10. În informatică, însă, sunt utilizate și alte sisteme de numerație, precum sistemul binar (baza 2), sistemul octal (baza 8) și sistemul hexazecimal (baza 16).

Bazele de numerație sunt importante în informatică deoarece calculatoarele *încă* prelucrează informația folosind circuite electronice care pot reprezenta două stări distincte, asociate în mod obișnuit valorilor 0 și 1. Din acest motiv, sistemul binar ocupă un rol fundamental în reprezentarea și prelucrarea datelor.

Un număr natural **N**, scris într-un sistem de numerație pozițional cu baza **b,** are forma generală:

$$(N)_b = a_k b^k + a_{k-1} b^{k-1} + \cdots + a_1 b + a_0 b^0$$

unde:

- b este baza sistemului de numerație (b>=2);

- $a_0, a_1, \ldots, a_k$ sunt cifrele numărului, cu $0 <= a_i < b$;

- a_i reprezintă cifra de pe poziția i, numerotând pozițiile de la dreapta la stânga, începând cu poziția 0. Exemplu:

$$(231)_4 = 2 \cdot 4^2 + 3 \cdot 4^1 + 1 \cdot 4^0$$

$$= 32 + 12 + 1 = (45)_{10}$$

2. Baza unui sistem de numerație

Baza unui sistem de numerație reprezintă numărul de simboluri distincte utilizate pentru scrierea numerelor în acel sistem pozițional (valoarea unei cifre depinde de poziția sa în număr).

> *Observație: un sistem nepozițional este sistemul roman de numerație* ☺

Într-un sistem de bază (b), cifrele posibile sunt cuprinse între 0 și (b-1).

De exemplu:

- baza 2 (binară): utilizează cifrele 0 și 1.
- baza 8 (octală): utilizează cifrele de la 0 la 7.
- baza 10 (zecimală): utilizează cifrele de la 0 la 9.
- baza 16 (hexazecimală): utilizează cifrele de la 0 la 9 și literele A, B, C, D, E și F, care reprezintă valorile de la 10 la 15.

În general, pentru o bază (b), fiecare cifră trebuie să aibă o valoare mai mică decât baza respectivă.

3. Conversia între baze de numerație

Conversia între două baze de numerație presupune reprezentarea aceluiași număr folosind simbolurile și regulile bazei de destinație, fără modificarea valorii numerice.

Există două situații principale.

a) Conversia dintr-o bază oarecare în baza 10

Se utilizează descompunerea numărului în suma produselor dintre cifre și puterile bazei.

b) Conversia din baza 10 într-o altă bază

Pentru numerele întregi nenegative se efectuează împărțiri succesive la baza de destinație. Resturile obținute se citesc în ordine inversă, de la ultimul la primul.

De exemplu, pentru conversia lui (25) din baza 10 în baza 2, împărțim succesiv la 2 și obținem resturile 1, 0, 0, 1, 1. Citite invers, acestea formează numărul $(11001)_2$.

! În cazul bazelor de forma (2^k), conversia se poate realiza mai rapid prin gruparea cifrelor binare. De exemplu, trecerea din baza 2 în baza 8 se realizează prin gruparea biților câte trei, iar trecerea în baza 16 prin gruparea lor câte patru.
Trecerea lui n din baza p în baza q se face scriind n în baza 10 și apoi efectuând conversia în baza q!

4. Rolul algoritmilor în prelucrarea numerelor

Prin utilizarea operațiilor aritmetice și a structurilor repetitive, putem construi algoritmi pentru:

- transformarea numerelor dintr-o bază în alta;
- extragerea și prelucrarea cifrelor unui număr;
- reprezentarea numerelor în sisteme de numerație diferite;
- efectuarea rapidă a conversiilor între baze de forma 2^k;
- aplicarea unor metode alternative de calcul, precum înmulțirea à la Russe;
- reprezentarea numerelor folosind șirul Fibonacci.

În continuare, fiecare dintre aceste probleme poate fi studiată separat, prin prezentarea principiului matematic, a exemplelor de calcul și a pseudocodului/codului corespunzător.

5. Scrierea numerelor în baza 2

Sistemul de numerație binar (baza 2) folosește doar două cifre: 0 și 1. Fiecare cifră binară se numește bit.

În baza 10, valoarea unui număr se determină folosind puterile lui 10. În mod asemănător, în baza 2, fiecare cifră se înmulțește cu puterea corespunzătoare a lui 2.

De exemplu:

$$(1101)_2 = 1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0$$

$$(1101)_2 = (13)_{10}$$

Transformarea unui număr din baza 10 în baza 2

Pentru a transforma un număr natural din baza 10 în baza 2, efectuăm împărțiri succesive la 2 și reținem resturile. Numărul binar se obține citind resturile în ordine inversă, de la ultimul la primul.

Exemplu: transformarea numărului 13 în baza 2.

Prin urmare, $13_{10}=(1101)_2$.

```
 13 | 2
-12 +---
---   6 | 2
 (1) -6 +---
     ---   3 | 2
      (0) -2 +---
          ---   1 | 2
           (1) -0 +---
               ---   0
                (1)
                 ▲
                 | Citim resturile
                 | de jos în sus: 1 1 0 1
```

Pseudocod

```
citește n
i ← 0
cât timp n > 0 execută
    r ← n % 2
    i ← i + 1
    v[i] ← r
    n ← [n / 2]
sfârșit_cât_timp
dacă i = 0 atunci
    scrie 0
altfel
    cât timp i > 0 execută
        scrie v[i]
        i ← i - 1
    sfârșit_cât_timp
sfârșit_dacă
```

Explicații:

- `n % 2` calculează restul împărțirii lui `n` la 2.
- `[n / 2]` reprezintă câtul întreg al împărțirii.
- Resturile se memorează în vectorul `v`, deoarece trebuie afișate în ordine inversă.
- Dacă `n = 0`, rezultatul este `0`, întrucât bucla nu se execută.

C++

```
//Varianta cu vector din STL
#include <iostream>
#include <vector>
using namespace std;

int main() {
    int n;
    cout << "n= ";
    cin >> n;
    int i = 0;
    vector<int> v;
    // vectorul v este gol
    //se folosește STL
    while (n > 0)
    {
        int r = n % 2;
        i = i + 1;
        v.push_back(r);
        // se adauga un element
        n = n / 2;
    }
    if (i == 0)
        cout << 0 << endl;
    else {
        while (i > 0) {
            cout << v[i - 1];
            i = i - 1;
        }
        cout << endl;
    }
    return 0;
}
//Varianta cu vector, fără STL
#include <iostream>
using namespace std;

int main()
{
    int n;
    cout << "n= ";
```

```
    cin >> n;
    int i = 0;
    int v[1000];
    while (n > 0)
    {
        int r = n % 2;
        i = i + 1;
        v[i - 1] = r;
        n = n / 2;
    }
    if (i == 0)
        cout << 0 << endl;
    else
    {
        while (i > 0)
        {
            cout << v[i - 1];
            i = i - 1;
        }
        cout << endl;
    }

    return 0;
}
```

Explicații:

Se folosește biblioteca *vector* pentru operațiile cu vectori în STL, în prima variantă.

Transformarea unui număr din baza 2 în baza 10

Pentru conversia din baza 2 în baza 10, putem prelucra cifrele de la dreapta la stânga. Fiecare cifră contribuie la rezultat în funcție de poziția sa.

De exemplu:

$$(1011)_2 = 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 = 11$$

Pseudocod	C++

citește n
putere ← 1
s ← 0
execută
    cif ← n % 10
    s ← s + cif * putere
    putere ← putere * 2
    n ← [n / 10]
cât timp n ≠ 0
scrie s
	#include <iostream>
	using namespace std;
	
	int main() {
	    int n;
	    cout << "n = ";
	    cin >> n;
	    int putere = 1;
	    int s = 0;
	    while (true)
	    {
	        int cif = n % 10;
	        s = s + cif * putere;
	        putere = putere * 2;
	        n = n / 10;
	        if (n == 0)
	            break;
	    }
	    cout << s << endl;
	
	    return 0;
	}
	
	Observație: în acest algoritm, numărul binar este citit ca un număr zecimal format din cifrele 0 și 1. Instrucțiunea repetitivă cu test final permite procesarea corectă și a numărului 0.

6. Scrierea numerelor în baza 16

Sistemul de numerație hexazecimal (baza 16) utilizează 16 simboluri:

Valoare zecimală	Cifră hexazecimală
0–9	0–9
10	**A**
11	**B**
12	**C**
13	**D**
14	**E**
15	**F**

De exemplu:

$$(2AF)_{16} = 2 \cdot 16^2 + 10 \cdot 16 + 15 = 687$$

Transformarea unui număr din baza 10 în baza 16

Algoritmul este asemănător celui pentru baza 2, dar împărțirile succesive se fac la 16. Resturile sunt cuprinse între 0 și 15, iar valorile de la 10 la 15 se reprezintă prin literele A–F.

Pseudocod

```
citește n
i ← 0

cât timp n > 0 execută
    r ← n % 16
    i ← i + 1
    dacă r < 10 atunci
        v[i] ← r
    altfel
        v[i] ← caracterul
corespunzător valorii r
    sfârșit_dacă
    n ← [n / 16]
sfârșit_cât_timp

dacă i = 0 atunci
    scrie 0
altfel
    cât timp i > 0 execută
```

C++

```
#include <iostream>
using namespace std;

int main() {
    int n;
    cout << "n= ";
    cin >> n;

    int i = 0;
    char v[1000];
    char simboluri[] = "0123456789ABCDEF";

    while (n > 0)
    {
        int r = n % 16;
        i = i + 1;
        v[i - 1] = simboluri[r];
        n = n / 16;
    }
```

```
        scrie v[i]
        i ← i - 1
    sfârșit_cât_timp
sfârșit_dacă
```

În acest pseudocod, instrucțiunea „caracterul corespunzător valorii r” desemnează asocierea 10 → A, 11 → B, ..., 15 → F.
Exemplu: pentru n = 687, resturile obținute sunt 15, 10, 2. Citite invers, ele formează 2AF.

```
    if (i == 0)
        cout << 0;
    else
    {
        while (i > 0)
        {
            cout << v[i - 1];
            i = i - 1;
        }
    }

    return 0;
}
```

Explicații pe scurt:
simboluri[r] –șir de caractere necesar pentru a obține cifra hexazecimală corespunzătoare restului (0–9, A–F).
v[i - 1] = simboluri[r] memorează cifra în tablou.

Transformarea unui număr din baza 16 în baza 10

Pentru a transforma un număr hexazecimal în baza 10, înmulțim fiecare cifră cu puterea corespunzătoare a lui 16 și adunăm rezultatele.

$$(2AF)_{16} = 2 \cdot 16^2 + 10 \cdot 16 + 15 = 687$$

Pseudocod	C++

citește n
putere ← 1
s ← 0
execută
cif ← n % 10
dacă cif < 10 atunci
valoare ← cif
altfel
valoare ← cifră hexazecimală convertită în valoare
sfârșit_dacă
s ← s + valoare * putere
	#include <iostream>
	#include <cstring>
	using namespace std;
	
	int main()
	{
	char n[1000];
	cout << "n= ";
	cin >> n;
	int i = 0;
	int s = 0;
	int putere = 1;

```
    putere ← putere * 16
    n ← [n / 10]
cât timp n ≠ 0
scrie s
```

Important: pseudocodul de mai sus este schematic pentru cifrele hexazecimale, deoarece un număr care conține litere precum A, B sau F nu poate fi prelucrat direct prin operații aritmetice asupra unui număr întreg. Într-o implementare completă, se citește un șir de caractere și se transformă fiecare caracter în valoarea sa numerică.

```
    char simboluri[] = "0123456789ABCDEF";
    while (i < strlen(n))
    {
        int cif = 0;
        while (simboluri[cif] != n[strlen(n) - 1 - i])
        {
            cif = cif + 1;
        }
        s = s + cif * putere;
        putere = putere * 16;
        i = i + 1;
    }
    cout << s;

    return 0;
}
```

Bucla interioară caută caracterul în șirul simboluri. Poziția la care îl găsește este chiar valoarea cifrei hexazecimale. De exemplu, pentru F, programul găsește caracterul la poziția 15, deci obține cif = 15. Apoi poate calcula contribuția cifrei la rezultat.

7. Baza Fibonacci

Baza Fibonacci este un sistem de reprezentare a numerelor naturale folosind numerele din șirul Fibonacci.

Șirul Fibonacci este definit astfel:

$$F_1 = 1, \quad F_2 = 2, \quad F_k = F_{k-1} + F_{k-2}$$

Astfel, termenii utilizați sunt:

$$1,\ 2,\ 3,\ 5,\ 8,\ 13,\ 21,\ 34,\ 55, \ldots$$

> *Observație: în această convenție, un număr natural se reprezintă ca o sumă de termeni Fibonacci distincți. Pentru a obține o reprezentare unică, nu se folosesc două cifre 1 consecutive.*

De exemplu:

$$19=13+5+1$$

Prin urmare, reprezentarea în baza Fibonacci, începând cu termenii 13,8,5,3,2,1 este:

$$(101001)_F$$

Verificare:

$$1 \cdot 13 + 0 \cdot 8 + 1 \cdot 5 + 0 \cdot 3 + 0 \cdot 2 + 1 \cdot 1 = 19$$

Transformarea unui număr natural în baza Fibonacci

Metoda aplicată este Greedy: determinăm cel mai mare termen Fibonacci care nu depășește numărul și îl scădem. Repetăm procedeul cu diferența obținută.

Pseudocod	C++

citește n
F[1] ← 1
F[2] ← 2
k ← 2
cât timp F[k] ≤ n execută
    k ← k + 1
    F[k] ← F[k - 1] + F[k - 2]
sfârșit_cât_timp
k ← k - 1
cât timp k ≥ 1 execută
    dacă n ≥ F[k] atunci
        scrie 1
        n ← n - F[k]
    altfel
        scrie 0
    sfârșit_dacă
    k ← k - 1
sfârșit_cât_timp
	#include <iostream>
	using namespace std;
	
	int main() {
	    int n;
	    cout << "n= ";
	    cin >> n;
	    /*Vectorul F este utilizat pentru memorarea termenilor din șirul lui Fibonacci*/
	    int F[1000] = {0, 1, 2};
	    int k = 2;
	    while (F[k] <= n)
	    {
	        k = k + 1;
	        F[k] = F[k - 1] + F[k - 2];
	    }
	    k = k - 1;
	    while (k >= 1)
	    {
	        if (n >= F[k])

	{
	cout << 1;
	n = n - F[k];
	}
	else
	cout << 0;
	k = k - 1;
	}
	return 0;
	}
	
	Explicații:
	- Se construiește șirul Fibonacci până la primul termen mai mare decât n.
	- Se parcurg termenii în ordine descrescătoare.
	- Dacă termenul curent poate fi scăzut din n, se afișează 1 și se scade valoarea sa.
	- În caz contrar, se afișează 0.
	Pentru n = 19, algoritmul obține 101001.
	Pentru n = 0, este necesară tratarea separată a cazului inițial, astfel încât rezultatul să fie 0, nu o secvență de cifre.

Transformarea din baza Fibonacci în baza 10

Pentru conversie, fiecare cifră se înmulțește cu termenul Fibonacci corespunzător poziției sale, apoi valorile se adună.

De exemplu:

$$(101001)_F = 1 \cdot 13 + 0 \cdot 8 + 1 \cdot 5 + 0 \cdot 3 + 0 \cdot 2 + 1 \cdot 1 = 19$$

Pseudocod	C++
citește n // n conține cifrele reprezentării Fibonacci citește numărul de cifre m F[1] ← 1 F[2] ← 2	#include <iostream> #include <cstring> using namespace std; int main() { char n[1000];

```
pentru i ← 3, m execută
    F[i] ← F[i - 1] + F[i - 2]
sfârșit_pentru
s ← 0
putere ← m

pentru i ← 1, m execută
    cif ← cifra de pe poziția
i din n
    s ← s + cif * F[putere]
    putere ← putere - 1
sfârșit_pentru
scrie s
```

```
    // numarul este un sir de caractere
de forma 101001
    cout << "n= ";
    cin >> n;
    int m = strlen(n);
    int F[1001] = {0};
    //vectorul   F   retine   termenii
Fibonacci
    F[1] = 1;
    if (m >= 2)
        F[2] = 2;
    for (int i = 3; i <= m; i++)
        F[i] = F[i - 1] + F[i - 2];
    int s = 0;
    int putere = m;
    for (int i = 0; i < m; i++)
    {
        int cif = n[i] - '0';
        s = s + cif * F[putere];
        putere = putere - 1;
    }
    cout << s;
    return 0;
}
```

8. Înmulțirea à la russe (Metoda înjumătățirii și dublării)

Denumiri alternative:

1. Înmulțirea țărănească rusească (*Russian peasant multiplication*) – Numele provine din literatura anglo-saxonă de la începutul secolului XX, unde s-a consemnat că metoda era folosită frecvent de țăranii ruși care nu cunoșteau tabla înmulțirii clasice.
2. Înmulțirea egipteană (sau antică egipteană) – Din punct de vedere istoric, algoritmul este mult mai vechi decât Rusia; metoda dublării și înjumătățirii succesive apare descrisă pe papirusurile matematice din Egiptul Antic (cum ar fi Papirusul Matematic Rhind, copiat de scribul Ahmes în jurul anului 1550 î.Hr.).
3. Înmulțirea etiopiană – O variantă a aceleiași metode a fost documentată istoric ca fiind folosită în mod tradițional în Etiopia.
4. Metoda înjumătățirii și dublării (în engleză: mediation and duplation) – Este denumirea tehnică, strict matematică, care descrie exact cele două operații de bază folosite în algoritm.

Înmulțirea à la russe este o metodă de înmulțire bazată pe împărțiri succesive la 2 și dublări succesive.

Exemplu: 13*12:

Scrierea lui 13 în baza 2: $13_{10} = 1101_2$

Asta înseamnă că $13 = 1*2^3 + 1*2^2 + 0*2^1 + 1*2^0 = 2^3 + 2^2 + 2^0$

$13*12 = (2^3 + 2^2 + 2^0) * 12 = 12*2^3 + 12*2^2 + 12*2^0$. În tabelul următor sunt descrise operațiile care contribuie la obținerea sumei de termeni de forma $k*2^x$.

Primul număr	Al doilea număr	Se adună la sumă?
13	$12*2^0$	Da
6	$12*2^1$	Nu
3	$12*2^2$	Da
1	$12*2^3$	Da
0	$12*2^4$	Stop

Rezultatul este:

$$13 \cdot 12 = 12 + 48 + 96 = 156$$

Pseudocod

```
citește a, b
p ← 0
cât timp a > 0 execută
    dacă a % 2 = 1 atunci
        p ← p + b
    sfârșit_dacă
    a ← [a / 2]
    b ← b * 2
sfârșit_cât_timp
scrie p
```

Explicații:
- a % 2 = 1 verifică dacă primul număr este impar.

C++

```
#include <iostream>
using namespace std;

int main()
{
    int a, b;
    cout << "a= ";
    cin >> a;
    cout << "b= ";
    cin >> b;
    int p = 0;
    while (a > 0)
    {
        if (a % 2 == 1)
            p = p + b;
        a = a / 2;
```

- Dacă este impar, valoarea b se adaugă la produsul parțial p.
- a ← [a / 2] înjumătățește primul număr.
- b ← b * 2 dublează al doilea număr.
Algoritmul funcționează pentru numere naturale nenegative. Dacă unul dintre factori este 0, produsul este 0.
	`b = b * 2;`
	`}`
	`cout << p;`
	`return 0;`
	`}`

9. Trecerea rapidă între baze de forma 2^k

Bazele de forma 2^k sunt bazele care reprezintă puteri ale lui 2. Conversiile între aceste baze sunt rapide deoarece fiecare cifră dintr-o astfel de bază corespunde unui număr fix de biți.

Baza	Valoarea lui k	Numărul de biți grupați
2	1	1 bit
4	2	2 biți
8	3	3 biți
16	4	4 biți
32	5	5 biți

Trecerea din baza 2 în baza 8

Pentru a transforma un număr binar în baza 8, grupăm cifrele binare câte trei, începând din dreapta. Dacă grupul din stânga nu conține trei cifre, completăm cu zerouri la stânga.

Exemplu:

$$(1101011)_2 = (001\ 101\ 011)_2$$

Fiecare grup se transformă într-o cifră octală:

- 001 → 1
- 101 → 5
- 011 → 3

Rezultă:

$$\boxed{(1101011)_2 = (153)_8}$$

Pseudocod

```
citește șirul binar s
m ← numărul de cifre din s
cât timp m % 3 ≠ 0 execută
    adaugă un 0 la începutul șirului s
    m ← m + 1
sfârșit_cât_timp
împarte s în grupe de câte 3 cifre

pentru fiecare grup g execută
    valoare ← valoarea binară a grupului g
    scrie valoare
sfârșit_pentru
```

C++

```
#include <iostream>
#include <cstring>
using namespace std;

int main()
{
    char s[1000];
    cout << "Citeste sirul binar: ";
    cin >> s;
    int m = strlen(s);
    while (m % 3 != 0)
    {
        for (int i = m; i >= 0; i--)
            s[i + 1] = s[i];
        s[0] = '0';
        m = m + 1;
    }
    for (int i = 0; i < m; i = i + 3)
    {
        int valoare = 0;
        for (int j = 0; j < 3; j++)
            valoare = valoare * 2 + (s[i + j] - '0');
        cout << valoare;
    }
    return 0;
}
```

_____

Explicații:

- strlen(s) calculează numărul de cifre din șir.
- `while (m % 3 != 0)` adaugă zerouri la început până când lungimea șirului este divizibilă cu 3.
- `for (int j = 0; j < 3; j++) valoare = valoare * 2 + (s[i + j] - '0');` extrage câte un grup de 3 cifre.
- afișează cifrele una lângă alta, obținând numărul în baza 8.

Trecerea din baza 2 în baza 16

Pentru conversia din baza 2 în baza 16, grupăm cifrele binare câte patru, tot de la dreapta la stânga. Dacă este necesar, completăm grupul din stânga cu zerouri.

Exemplu:

$$(110101101)_2 = (0001\ 1010\ 1101)_2$$

Grupele se transformă astfel:

- 0001 → 1
- 1010 → A
- 1101 → D

Prin urmare:

$$\boxed{(110101101)_2 = (1AD)_{16}}$$

Pseudocod	C++

citește șirul binar s
m ← numărul de cifre din s
cât timp m % 4 ≠ 0 execută
    adaugă un 0 la începutul șirului s
    m ← m + 1
sfârșit_cât_timp
 	
împarte s în grupe de câte 4 cifre
 	
pentru fiecare grup g execută
    valoare ← valoarea binară a grupului g
 	
    dacă valoare < 10 atunci
        scrie valoare
    altfel
        scrie caracterul corespunzător valorii
    sfârșit_dacă
	#include <iostream>
	#include <cstring>
	using namespace std;
	
	int main()
	{
	    char s[1000];
	    cout << "Citeste sirul binar: ";
	    cin >> s;
	    int m = strlen(s);
	    while (m % 4 != 0)
	    {
	        for (int i = m; i >= 0; i--)
	            s[i + 1] = s[i];
	        s[0] = '0';
	        m = m + 1;
	    }
	    char simboluri[] = "0123456789ABCDEF";
	    for (int i = 0; i < m; i = i + 4)
	    {
	        int valoare = 0;

```
sfârșit_pentru
```

```
        for (int j = 0; j < 4; j++)
            valoare = valoare * 2 + (s[i + j] - '0');
        if (valoare < 10)
            cout << valoare;
        else
            cout << simboluri[valoare];
    }
    return 0;
}
```

Explicații:

- strlen(s) calculează numărul de cifre din șir.
- `while (m % 4 != 0)` adaugă zerouri la început până când lungimea este multiplu de 4.
-
```
for (int j = 0; j < 4; j++)
  valoare = valoare * 2 + (s[i + j] - '0');
    if (valoare < 10)
        cout << valoare;
    else
        cout << simboluri[valoare]
```
extrage câte un grup de 4 cifre.
- `simboluri[valoare]` obține caracterul hexazecimal corespunzător: `10 → A, 11 → B, …, 15 → F.`

Trecerea din baza 16 în baza 2

Conversia se realizează prin înlocuirea fiecărei cifre hexazecimale cu reprezentarea sa binară pe exact patru biți.

Hexazecimal	Binar	Hexazecimal	Binar
0	0000	8	1000
1	0001	9	1001
2	0010	A	1010
3	0011	B	1011
4	0100	C	1100
5	0101	D	1101
6	0110	E	1110
7	0111	F	1111

Exemplu:

$$(3B7)_{16} = (0011\ 1011\ 0111)_2$$

Eliminând zerourile nesemnificative din stânga, obținem:

$$\boxed{(3B7)_{16} = (1110110111)_2}$$

Pseudocod	C++

citește șirul hexazecimal s
 	
pentru fiecare caracter c din s execută
valoare ← valoarea cifrei hexazecimale c
scrie reprezentarea binară a valorii pe 4 biți
sfârșit_pentru
	#include <iostream>
	#include <cstring>
	using namespace std;
	
	int main() {
	char s[1000];
	cout << "Citeste sirul hexazecimal: ";
	cin >> s;
	char simboluri[] = "0123456789ABCDEF";
	for (int i = 0; i < strlen(s); i++)
	{
	char c = s[i];
	if (c >= 'a' && c <= 'f')
	c = c - 'a' + 'A';
	int valoare = 0;
	while (simboluri[valoare] != c)
	valoare++;
	int binar[4];
	for (int j = 3; j >= 0; j--)
	{
	binar[j] = valoare % 2;
	valoare = valoare / 2;
	}
	for (int j = 0; j < 4; j++)
	cout << binar[j];
	}
	return 0;
	}
	
	_____
	Explicații:
	• `char s[1000]` memorează șirul hexazecimal introdus.

	• `simboluri` conține caracterele `0-9` și `A-F`.
	• Dacă o literă este mică, de exemplu `f`, o transformăm în literă mare, `F`.
	• Bucla `while (simboluri[valoare] != c)` caută caracterul în șirul `simboluri`. Poziția găsită reprezintă valoarea hexazecimală.
	• Tabloul `binar[4]` memorează cele patru cifre binare ale valorii.
	• `valoare % 2` determină ultima cifră binară, iar `valoare / 2` elimină acea cifră.
	• Cifrele sunt afișate de la poziția `0` la `3`, pentru a păstra ordinea corectă.
	Exemplu:
	`Citeste sirul hexazecimal: 2F`
	`00101111`