INFORMATICĂ · CLASA A IX-A · ALGORITMI ELEMENTARI · LECȚIA 13

Baze de numerație: baza 2, baza 10, baza b

Durată: 60 min Nivel: toate profilurile Înainte: lecțiile ix-09

CE VEI ȘTI DUPĂ LECȚIE

- Înțelegi ce înseamnă scrierea unui număr într-o bază b.
- Transformi un număr din baza 10 în baza b (împărțiri repetate) și invers.
- Faci conversii rapide 2 ↔ 8 ↔ 16 pe hârtie.
- Rezolvi probleme cu cifrele binare ale unui număr.

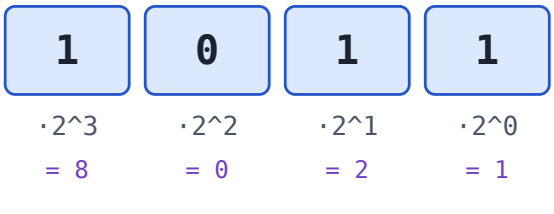
1. Ce înseamnă o bază de numerație

În baza 10 avem cifrele 0–9, iar poziția unei cifre îi dă valoarea:

$3507 = 3 \cdot 10^3 + 5 \cdot 10^2 + 0 \cdot 10^1 + 7 \cdot 10^0$

În **baza b** avem cifrele 0 ... b−1, iar pozițiile valorează puteri ale lui b:

$1011_{(2)} = 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 = 8 + 0 + 2 + 1 = \mathbf{11}$



1011 în baza 2

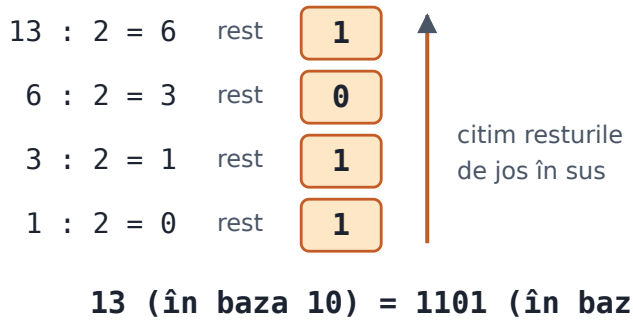
Baza	Cifre	Unde apare
2 (binar)	0, 1	memoria calculatorului (biți)
8 (octal)	0–7	permisiuni de fișiere în Linux
10 (zecimal)	0–9	viața de zi cu zi
16 (hexazecimal)	0–9, A–F	culori (#FF8800), adrese de memorie

DE CE AȘA?

De ce calculatoarele folosesc baza 2? Un circuit electronic are două stări sigure: curent / fără curent. Zece stări distincte ar fi scumpe și nesigure, două sunt ieftine și robuste. Orice număr, text, imagine sau sunet se reduce la un șir de 0 și 1; baza 16 e doar o scriere scurtă pentru cititorul uman.

2. Din baza 10 în baza b: împărțiri repetate

Împărțim repetat la b; **resturile, citite de jos în sus**, sunt cifrele.



13 în baza 2

13 în baza 2: 13 : 2 = 6 rest **1** → 6 : 2 = 3 rest **0** → 3 : 2 = 1 rest **1** → 1 : 2 = 0 rest **1** → 13 = $\mathbf{1101_{(2)}}$

DE CE AȘA?

De ce funcționează? Restul la împărțirea cu b este ultima cifră în baza b (exact cum `n % 10` e ultima cifră în baza 10). Împărțirea `n / b` „taie" ultima cifră. Algoritmul de la cifrele unui număr, aplicat cu b în loc de 10. Resturile ies în ordine inversă: de la ultima cifră la prima, de aceea le citim de jos în sus.

În cod, resturile vin în ordine inversă, așa că folosim metoda cu puterea lui 10 (ca la construirea numerelor din cifre):

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

int main()
{
    long long n, b, rez = 0, p = 1;
    cin >> n >> b;
    while (n > 0)
    {
        rez += (n % b) * p;
        p *= 10;
        n /= b;
    }
    cout << rez;
    return 0;
}
```

n	`n % 2`	p	rez	n după
13	1	1	1	6
6	0	10	1	3
3	1	100	101	1

n	`n % 2`	p	rez	n după
1	1	1000	1101	0

> **ATENȚIE — GREȘELI FRECVENTE**
>
> Metoda funcționează doar pentru b ≤ 10 și numere mici (un număr de 10 cifre în baza 2 are ~30 de cifre și nu mai încape într-un `long long` ca „număr zecimal”). Pentru baze mari sau numere mari, memorează cifrele într-un vector sau afișează-le recursiv.

3. Din baza b în baza 10

Metoda 1 (puteri): parcurgi cifrele de la dreapta și aduni `cifra · b^k`:

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

int main()
{
    long long x, b, val = 0, p = 1;     // x = cifrele în baza b, scrise „ca un număr zecimal”
    cin >> x >> b;
    while (x > 0)
    {
        val += (x % 10) * p;
        p *= b;
        x /= 10;
    }
    cout << val;
    return 0;
}
```

x	cifra `x % 10`	p	val
1101	1	1	1
110	0	2	1
11	1	4	5
1	1	8	13

Metoda 2 (Horner), când citești cifrele de la stânga la dreapta:

```cpp
val = 0;
// pentru fiecare cifră c, în ordinea de la stânga:
val = val * b + c;
```

Exemplu: $1101_{(2)}$ → 0 → 1 → 1·2+1 = 3 → 3·2+0 = 6 → 6·2+1 = **13**.

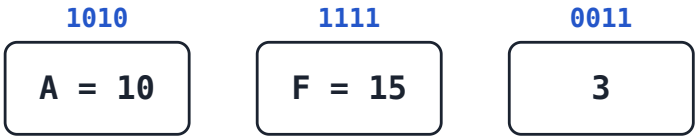
DE CE AȘA?

De ce funcționează Horner? Fiecare cifră nouă „împinge" tot ce ai până acum cu o poziție spre stânga (înmulțirea cu b) și intră la unități. La baza 10 faci același lucru când citești un număr cifră cu cifră: din „12" și cifra 3 obții 123 = 12·10 + 3. E exact `og = og * 10 + c` de la numărul oglindit.

4. Conversii rapide între bazele 2, 8, 16

Fiindcă 8 = 2^3 și 16 = 2^4:

- o cifră octală ↔ **3 biți**: $101\ 110_{(2)} = 56_{(8)}$;
- o cifră hexazecimală ↔ **4 biți**: $1010\ 1111_{(2)} = AF_{(16)}$.



1010 1111 0011 (baza 2) = AF3 (baza 16)

Grupe de 4 biți

Hex	0	1	...	9	A	B	C	D	E	F
Zecimal	0	1	...	9	10	11	12	13	14	15
Binar	0000	0001	...	1001	1010	1011	1100	1101	1110	1111

DE CE AȘA?

De ce hexazecimal în loc de zecimal pentru culori? Un canal de culoare are 8 biți = 2 cifre hexazecimale exact (00–FF = 0–255). Culoarea #FF8800 înseamnă roșu 255, verde 136, albastru 0: fiecare pereche de cifre e un octet. În zecimal nu există o împărțire atât de curată.

SFAT

Ca să convertești orice număr zecimal în binar „în cap", scrie puterile lui 2 (1, 2, 4, 8, 16, 32, 64, 128) și scade-le de la cea mai mare care încape: 100 = 64 + 32 + 4 → $1100100_{(2)}$.

5. Biții unui număr (bonus)

- `n % 2` = ultimul bit, `n / 2` = taie ultimul bit (exact ca la cifrele zecimale).
- Puterile lui 2 au **un singur bit de 1**: 1, 10, 100, 1000...$_{(2)}$.
- Operatorii pe biți `&`, `|`, `^`, `<<`, `>>` lucrează direct pe biți: `1 << k` = 2^k, `n & 1` = ultimul bit.

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

int main()
{
    int n, uni = 0;
    cin >> n;
    while (n > 0)
    {
        uni += n % 2;      // același lucru cu n & 1
        n /= 2;
    }
    cout << uni;           // câți biți de 1 are n
    return 0;
}
```

6. Greșeli frecvente

Greșeala	Efect	Corect
resturile citite de sus în jos	numărul oglindit	de jos în sus
`val = val + c * b` la Horner	rezultat greșit	`val = val * b + c`
cifră ≥ baza (de exemplu `2` în binar)	număr invalid	verifică cifrele
scrii `1101` și crezi că e „o mie o sută unu”	confuzie între baze	notează baza: $1101_{(2)}$
`p` pornit de la 0	rezultat 0	`p = 1`

7. La BAC

LA BAC

Bazele de numerație nu mai apar în grilele actuale de la BAC (mate-info), dar ideea lor e peste tot: algoritmul „împărțiri repetate” e algoritmul cifrelor, iar grilele cu `n % 2` și `n / 2` sunt, de fapt, grile despre baza 2 (de exemplu „câți biți de 1” sau „n este putere a lui 2”). La concursuri (olimpiade) bazele apar direct.

8. Exersează pe hârtie

Răspunsurile sunt la final: încearcă întâi singur.

1. Transformă în baza 2: 25, 64, 100.

2. Transformă în baza 10: $11111_{(2)}$, $777_{(8)}$, $FF_{(16)}$.

3. Care este cea mai mică bază în care poate fi scris numărul 4073? (Indiciu: cifra maximă.)

4. Câte numere naturale au exact 5 cifre în baza 2?

5. Scrie 45 în baza 2.

6. Convertește $AF_{(16)}$ în baza 10, trecând prin baza 2.

7. Câte cifre are 1000 în baza 2?

8. Ce afișează secvența?

```cpp
int n = 13, c = 0;
while (n > 0) { c += n % 2; n /= 2; }
cout << c;
```

9. Ce valoare au `1 << 5` și `13 & 1`?

10. Scrie un program care afișează `DA` dacă `n` este putere a lui 2 și `NU` altfel (indiciu: câți biți de 1 are o putere a lui 2?).

9. Mai departe (pentru cine vrea mai mult)

Numere negative. Calculatorul reprezintă `-5` prin „complement față de 2": se inversează toți biții lui 5 și se adaugă 1. Așa se face că `int` merge de la -2^{31} la $2^{31} - 1$ și că depășirea unui `int` „se întoarce" la valori negative.

De ce `0.1 + 0.2 != 0.3`. Numerele reale se păstrează în binar. Fracția 0,1 nu are o scriere finită în baza 2 (la fel cum 1/3 nu are o scriere finită în baza 10), deci se rotunjește. De aceea comparațiile cu `==` între `double` sunt riscante: compară cu o toleranță (`fabs(a - b) < 1e-9`).

Trucuri cu biți. `n & (n - 1)` șterge cel mai din dreapta bit de 1: un număr e putere a lui 2 dacă `n > 0 && (n & (n - 1)) == 0`. `n & 1` testează paritatea mai rapid decât `% 2`. Măști de biți (`1 << k`) reprezintă submulțimi: o mulțime de 20 de elemente încape într-un `int`.

Alte baze. Baza 3 echilibrată (cifre −1, 0, 1), baza fibonacci, baza −2: sunt subiecte de concurs. Principiul e același: împărțiri repetate cu atenție la rest.

Rezumat

- Baza 10 → baza b: împărțiri repetate la b, resturile citite invers.
- Baza b → baza 10: Σ cifră · $b^{\text{poziție}}$ sau Horner (`val = val * b + c`).
- 3 biți = o cifră octală, 4 biți = o cifră hexazecimală.
- `n % 2` și `n / 2` sunt „ultimul bit" și „taie ultimul bit".

Probleme de lucru (pbinfo)

Începe cu acestea, în ordine: sunt alese să urmeze pașii din lecție.

- [ ] **#429** **cifbin** — ușoară
  Câte cifre de 0 și de 1 are n în baza 2.
  https://www.pbinfo.ro/probleme/429/cifbin

- [ ] **#428** **transfb** — ușoară
  Din baza b în baza 10.
  https://www.pbinfo.ro/probleme/428/transfb

- [ ] #426 **bazab** — ușoară
  https://www.pbinfo.ro/probleme/426/bazab

Temă (pbinfo)

Trimite soluțiile pe pbinfo și notează punctajul obținut. Bifează ce ai terminat.

- [ ] #945 **baze** — ușoară
  Din baza b în baza c: treci prin baza 10.
  https://www.pbinfo.ro/probleme/945/baze
- [ ] #947 **baza16** — ușoară
  Cifrele peste 9 se scriu A–F.
  https://www.pbinfo.ro/probleme/947/baza16
- [ ] #427 **bazaminima** — ușoară
  https://www.pbinfo.ro/probleme/427/bazaminima
- [ ] #122 **cifrebinare** — ușoară
  https://www.pbinfo.ro/probleme/122/cifrebinare
- [ ] #2872 **easypow** — ușoară
  https://www.pbinfo.ro/probleme/2872/easypow

Pentru cei care vor mai mult

Probleme mai grele sau de concurs, pentru cine vrea să meargă dincolo de BAC.

- [ ] #3348 **sumaputeri2** — medie
  https://www.pbinfo.ro/probleme/3348/sumaputeri2
- [ ] #946 **douabaze** — dificilă
  https://www.pbinfo.ro/probleme/946/douabaze

Citește și exersează în plus

- Baze de numerație https://www.pbinfo.ro/articole/5562/baze-de-numeratie
- Operații pe biți https://www.pbinfo.ro/articole/62/operatii-pe-biti
- Binary Game — învață să numeri în binar — joc scurt pentru a citi rapid numere binare (engleză) https://catonmat.net/ftp/binary-game.html
- cppreference — operatori pe biți — tabelul complet &, |, ^, <<, >> (secțiunea Bitwise logic operators) https://en.cppreference.com/w/cpp/language/operator_arithmetic
- Python Tutor pentru C++ — urmărește cum se construiește rezultatul la conversia în altă bază https://pythontutor.com/cpp.html

Răspunsuri

Exersează pe hârtie

1. 25 = $11001_{(2)}$ (16 + 8 + 1); 64 = $1000000_{(2)}$; 100 = $1100100_{(2)}$ (64 + 32 + 4).

2. $11111_{(2)}$ = 31; $777_{(8)}$ = 7·64 + 7·8 + 7 = 511; $FF_{(16)}$ = 15·16 + 15 = 255.

3. Cifra maximă din 4073 este 7, deci baza minimă este **8** (într-o bază b cifrele sunt 0 … b − 1).

4. 16: numerele de la $10000_{(2)}$ = 16 până la $11111_{(2)}$ = 31.

5. 45 = 32 + 8 + 4 + 1 = **$101101_{(2)}$**.

6. AF = 1010 $1111_{(2)}$ = 128 + 32 + 8 + 4 + 2 + 1 = **175**; direct: 10·16 + 15 = 175.

7. 10 cifre: $2^9 = 512 \le 1000 < 1024 = 2^{10}$.

8. `3`. 13 = $1101_{(2)}$ are trei cifre de 1. Secvența numără biții de 1.

9. `1 << 5 = 32`; `13 & 1 = 1` (ultimul bit al lui 13 = $1101_{(2)}$).

10.

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

int main()
{
    int n, uni = 0;
    cin >> n;
    int m = n;
    while (m > 0)
    {
        uni += m % 2;
        m /= 2;
    }
    cout << (n > 0 && uni == 1 ? "DA" : "NU");
    return 0;
}
```

O putere a lui 2 are un singur bit de 1 (1, 10, 100…$_{(2)}$). Cazul `n = 0` are zero biți de 1 și nu e putere a lui 2.