Seminar 1
Biți

Baze de numerație

- Baza b < 10 -> cifrele {0, 1, … , b-1}

- Pentru exemplificare – baza b=2 (folosita și pentru reprezentarea internă a numerelor)

Baze de numerație

- Conversie baza 10 -> baza 2

Se dă un număr natural x în baza 10. Să se determine a1,..., an cifrele lui x în scrierea în baza 2:

$$x = \overline{a_1 \ldots a_n}_{(2)}$$

Baze de numerație

- Conversie baza 10 -> baza 2

Se dă un număr natural x în baza 10. Să se determine a1,..., an cifrele lui x în scrierea în baza 2:

$$x = \overline{a_1 \dots a_n}_{(2)}$$

$$x = a_n + a_{n-1} \cdot 2^1 + \cdots + a_1 \cdot 2^{n-1} \ \Rightarrow$$

Baze de numerație

- Conversie baza 10 -> baza 2

Se dă un număr natural x în baza 10. Să se determine a1,..., an cifrele lui x în scrierea în baza 2:

$$x = \overline{a_1 \ldots a_n}_{(2)}$$

$$x = a_n + a_{n-1} \cdot 2^1 + \cdots + a_1 \cdot 2^{n-1} \Rightarrow$$

$$a_n = x \, mod \, 2$$

$$a_{n-1} + a_{n-2} \cdot 2 \ldots + a_1 \cdot 2^{n-2} = x \, div \, 2 \Rightarrow$$

$$\Rightarrow a_{n-1} = x \, mod \, 2$$ etc

Baze de numerație

- Conversie baza 10 -> baza 2

Se dă un număr natural x în baza 10. Să se determine a1,..., an cifrele lui x în scrierea în baza 2:

$$x = \overline{a_1 \ldots a_n}_{(2)}$$

Idee algoritm: Împărțim succesiv la 2 și păstrăm resturile (obținem cifrele din reprezentarea în baza 2 de la dreapta la stânga)

Baze de numerație

- Conversie baza 10 -> baza 2

  Exemplu x= 86

Conversie baza 10 -> baza 2

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

int f10to2(unsigned int x) {
    int rez=0,p10=1,a;

    while(x>0){
       a=x%2;
       x=x/2;

       rez=rez+a*p10;
       p10*=10;
    }
    return rez;
}

//avem main
int main(){
    cout<<f10to2(235)<<endl;
}
```

Conversie baza 10 -> baza 2

<pre>#include<iostream>
using namespace std;
 	
int f10to2(unsigned int x) {
    int rez=0,p10=1,a;
 	
    while(x>0){
       a=x%2;
       x=x/2;
 	
       rez=rez+a*p10;
       p10*=10;
    }
    return rez;
}
 	
//avem main
int main(){
    cout<<f10to2(235)<<endl;
}</pre>
	<pre>#nu includem/importam nimic
	
	def f10to2(x):
	    rez=0
	    p10=1</pre>

Conversie baza 10 -> baza 2

<table>
<tr>
<td>

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

int f10to2(unsigned int x) {
    int rez=0,p10=1,a;

    while(x>0){
       a=x%2;
       x=x/2;

       rez=rez+a*p10;
       p10*=10;
    }
    return rez;
}

//avem main
int main(){
    cout<<f10to2(235)<<endl;
}
```

</td>
<td>

```python
#nu includem/importam nimic


def f10to2(x):
    rez=0
    p10=1
    while x>0:
       a=x%2
       x=x//2

       rez=rez+a*p10
       p10*=10

    return rez
```

</td>
</tr>
</table>

Conversie baza 10 -> baza 2

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

int f10to2(unsigned int x) {
    int rez=0,p10=1,a;

    while(x>0){
       a=x%2;
       x=x/2;

       rez=rez+a*p10;
       p10*=10;
    }
    return rez;
}

//avem main
int main(){
    cout<<f10to2(235)<<endl;
}
```

```python
#nu includem/importam nimic


def f10to2(x):
    rez=0
    p10=1
    while x>0:
       a=x%2
       x=x//2

       rez=rez+a*p10
       p10*=10

    return rez


#nu avem main
print(f10to2(235))
```

Baze de numerație

- Temă: Scrieți un program Python pentru conversia din baza 2 în 10

Reprezentarea numerelor întregi în calculator

- naturale – baza 2 $x = \overline{a_1 \dots a_n}_{(2)} \Rightarrow$ **$00\dots0a_1\dots a_n$**
- negative – complement față de 2, un bit pentru semn

$$x + (-x) = 0$$

$$\mathrm{repr}(x) + \mathrm{repr}(-x) = 2^n = 1\underbrace{0\ \dots\ 0}_{\text{n biți}}{}_{(2)}$$

Reprezentarea numerelor întregi în calculator

- naturale – baza 2 $x = \overline{a_1 \ldots a_n}_{(2)} \Rightarrow$ **$00\ldots0a_1\ldots a_n$**
- negative – complement față de 2, un bit pentru semn

$$x + (-x) = 0$$

$$repr(x) + repr(-x) = 2^n = 1\underbrace{0\ \ldots\ 0}_{n \text{ biți}}{}_{(2)}$$

not(repr(x)) = complementul lui repr(x)

$$repr(x) + not(repr(x)) = 1\ldots1_{(2)} = 2^n - 1$$

Reprezentarea numerelor întregi în calculator

- naturale – baza 2 $x = \overline{a_1 \ldots a_n}_{(2)} \Rightarrow$ **$00\ldots0a_1\ldots a_n$**
- negative – complement față de 2, un bit pentru semn

$$x + (-x) = 0$$

$$\mathrm{repr}(x) + \mathrm{repr}(-x) = 2^n = 1\underbrace{0 \ldots 0}_{\text{n biți}}{}_{(2)}$$

not(repr(x)) = complementul lui repr(x)

$$\mathrm{repr}(x) + \mathrm{not}(\mathrm{repr}(x)) = 1\ldots1_{(2)} = 2^n - 1$$

Avem atunci **repr(−x) = not(repr(x)) + 1 =**

= not(repr(x−1))

Reprezentarea numerelor întregi în calculator

- naturale – baza 2

$$x = \overline{a_1 \ldots a_n}_{(2)} \Rightarrow \text{repr}(x) = 00\ldots0a_1\ldots a_n$$

- negative – complement față de 2, un bit pentru semn

$$\text{repr}(-x) = \text{not}(\text{repr}(x)) + 1$$

Reprezentarea numerelor întregi

- **repr(–x) = not repr(x) + 1**

- **Exemplu:** pentru n=8 biți reprezentarea lui –11 se obține astfel:

repr(11) = 00001011

not(repr(11)) = 11110100

not(repr(11))+1 = 11110101 = repr(–11)

Operatori

- **Operatori pe biți**

–rapizi, asupra reprezentării interne

~x	complement față de 1
x & y	și pe biți
x | y	sau pe biți
x ^ y	sau exclusiv pe biți
x >> k	deplasare la dreapta cu k biți
x << k	deplasare la stânga cu k biți

^	0	1
0	0	1
1	1	0

Operatori pe biți

```
11 & 86 = ?

11 ^ 86 = ?
```

11:	0	0	0	0	1	0	1	1
86:	0	1	0	1	0	1	1	0
11 & 86:	0	0	0	0	0	0	1	0

Operatori pe biți

11 & 86 = ?

11 ^ 86 = ?

11:	0	0	0	0	1	0	1	1
86:	0	1	0	1	0	1	1	0
11 & 86:	0	0	0	0	0	0	1	0

→ 2

Operatori pe biți

11 & 86 = ?

11 ^ 86 = ?

11:	0	0	0	0	1	0	1	1
86:	0	1	0	1	0	1	1	0
11 ^ 86:	0	1	0	1	1	1	0	1

Operatori pe biți

```
11 & 86 = ?

11 ^ 86 = ?
```

11:	0	0	0	0	1	0	1	1	
86:	0	1	0	1	0	1	1	0	
11 ^ 86:	0	1	0	1	1	1	0	1	$\rightarrow$ 93

Operatori pe biți

~11 = ?

~0 = ?

11:	0	0	0	0	1	0	1	1	
y=~11:	1	1	1	1	0	1	0	0	Negativ; aflăm |y|

Operatori pe biți

~11 = ?

~0 = ?

11:	0	0	0	0	1	0	1	1	
y=~11:	1	1	1	1	0	1	0	0	Negativ; aflăm |y|
Scădem 1									
Trecem la complement									$\rightarrow$

Operatori pe biți

~11 = ?

~0 = ?

11:	0	0	0	0	1	0	1	1	
y=~11:	1	1	1	1	0	1	0	0	Negativ; aflăm |y|
Scădem 1	1	1	1	1	0	0	1	1	
Trecem la complement									

Operatori pe biți

`~11 = ?`

`~0 = ?`

11:	0	0	0	0	1	0	1	1	
y=~11:	1	1	1	1	0	1	0	0	Negativ; aflăm |y|
Scădem 1	1	1	1	1	0	0	1	1	
Trecem la complement	0	0	0	0	1	1	0	0	→ |y|=12

`~11 = -12`

(avem repr(–x) = not(repr(x–1)) => repr(–12)=not(repr(11)))

Operatori pe biți

```
11 >> 1 = ?
11 << 1 = ?
```

Operatori pe biți

11 >> 1 = 5

11 << 1 = 22

x >> k = parte întreagă inferioară x / 2^k (x div 2^k)

x << k = x * 2^k

Probleme

- Se dă un număr natural x. Să se calculeze x div 4 și x*6 folosind operatori pe biți

Probleme

- Se dă un număr natural x. Să se calculeze x div 4 și x*6 folosind operatori pe biți

```
print(x>>2)

print((x<<2) + (x<<1))
```

Probleme

- Se dă un număr întreg x. Să se determine dacă acesta este par sau impar folosind operatori pe biți.

Probleme

- Se dă un număr întreg x. Să se determine dacă acesta este par sau impar folosind operatori pe biți.

$x = \overline{a_1 \ldots an}_{(2)}$ par $\Leftrightarrow$ x mod 2 = 0 $\Leftrightarrow$ a_n =0

$\Rightarrow$ Trebuie să testăm dacă ultimul bit din reprezentarea lui x este 0

Probleme

- Se dă un număr întreg x. Să se determine dacă acesta este par sau impar folosind operatori pe biți.

$x = \overline{a_1 \ldots a_n}_{(2)}$ par $\Leftrightarrow$ x mod 2 = 0 $\Leftrightarrow$ $a_n = 0$

$\Rightarrow$ Trebuie să testăm dacă ultimul bit din reprezentarea lui x este 0:

x & 1 = a_n & 1 = a_n = ultimul bit

x	=	$a_1 \ldots \ a_{n-1} \ a_n$
1	=	$0 \ldots \ 0 \ 1$
x&1	=	$0 \ldots \ 0 \ a_n = a_n$

Probleme

- Se dă un număr întreg x. Să se determine dacă acesta este par sau impar folosind operatori pe biți.

```
#testam daca ultimul bit din reprezentarea binara este 0 sau 1
if x&1 == 0:
    print("par")
else:
    print("impar")
```

Probleme

- Se dau 2 numere naturale x și k. Să se afișeze al k-lea bit din dreapta din reprezentarea binară a lui x

Probleme

- Se dau 2 numere naturale x și k. Să se afișeze al k-lea bit din dreapta din reprezentarea binară a lui x

```
print( (x >> (k-1)) & 1)
```

Probleme

- Se citesc 2 numere întregi x și k. Să se afișeze numărul obținut din x astfel:
  - se setează bitul numărul k la valoarea 1
  - se setează bitul k la valoarea 0
  - se complementează bitul k

Probleme

- Se citesc 2 numere întregi x și k. Să se afișeze numărul obținut din x astfel:
  - se setează bitul numărul k la valoarea 1

Probleme

- Se citesc 2 numere întregi x și k. Să se afișeze numărul obținut din x astfel:
  - se setează bitul numărul k la valoarea 1

$$x \;=\; a_1 \ldots a_{n-k}\, a_{n-(k-1)}\, a_{n-(k-2)} \ldots a_n$$

$$m \;=$$

x op m

Probleme

- Se citesc 2 numere întregi x și k. Să se afișeze numărul obținut din x astfel:
  - se setează bitul numărul k la valoarea 1

$$
\begin{array}{lcl}
x & = & a_1 \ldots\ a_{n-k}\ a_{n-(k-1)}\ a_{n-(k-2)} \ldots\ a_n \\
m & = & 0. \ldots.\ 0 \quad\ 1 \quad\quad 0 \ldots\ldots.\ 0 \\
\hline
\color{red}{\mathbf{x|m}} & = & a_1 \ldots\ a_{n-k} \quad 1 \quad a_{n-(k-2)} \ldots\ a_n
\end{array}
$$

$m = ?$

Probleme

- Se citesc 2 numere întregi x și k. Să se afișeze numărul obținut din x astfel:
  - se setează bitul numărul k la valoarea 1

$$
\begin{array}{lcllll}
x & = & a_1 \ldots a_{n-k} & a_{n-(k-1)} & a_{n-(k-2)} \ldots a_n \\
m & = & 0. \ldots. 0 & 1 & 0 \ldots\ldots. 0 \\
\hline
\color{red}{\mathbf{x|m}} & = & a_1 \ldots a_{n-k} & 1 & a_{n-(k-2)} \ldots a_n
\end{array}
$$

$m = 2^{k-1} = (1 << (k-1))$

Probleme

- Se citesc 2 numere întregi x și k. Să se afișeze numărul obținut din x astfel:
  - se setează bitul numărul k la valoarea 1

```
print(x | (1 << (k-1)))
```

Probleme

- Se citesc 2 numere întregi x și k. Să se afișeze numărul obținut din x astfel:
  - se setează bitul numărul k la valoarea 0

$$x \quad = \; a_1 \ldots\, a_{n-k}\; a_{n-(k-1)}\; a_{n-(k-2)} \ldots\, a_n$$

$$m \;=$$

Probleme

- Se citesc 2 numere întregi x și k. Să se afișeze numărul obținut din x astfel:
  - se setează bitul numărul k la valoarea 0

$$x \quad = \; a_1 \ldots\, a_{n-k} \; a_{n-(k-1)} \; a_{n-(k-2)} \ldots\, a_n$$

$$m \quad = \; 1. \ldots.\, 1 \qquad 0 \qquad 1 \ldots\ldots.\, 1$$

$$\textcolor{red}{\mathbf{x\&m}} = a_1 \ldots\, a_{n-k} \qquad 0 \qquad a_{n-(k-2)} \ldots\, a_n$$

m = ?

Probleme

- Se citesc 2 numere întregi x și k. Să se afișeze numărul obținut din x astfel:
  - se setează bitul numărul k la valoarea 0

$$x \quad = \ a_1 \ldots a_{n-k} \ a_{n-(k-1)} \ a_{n-(k-2)} \ldots a_n$$

$$m \ = \ 1. \ldots. 1 \qquad 0 \qquad 1 \ldots\ldots. 1$$

$$\color{red}{\mathbf{x\&m}} = a_1 \ldots a_{n-k} \quad 0 \quad a_{n-(k-2)} \ldots a_n$$

$$m = \sim 2^{k-1} = \sim(1 << (k-1))$$

Probleme

- Se citesc 2 numere întregi x și k. Să se afișeze numărul obținut din x astfel:
  - se setează bitul numărul k la valoarea 0

```
print(x & (~ (1 << (k-1))))
```

Probleme

- Se citesc 2 numere întregi x și k. Să se afișeze numărul obținut din x astfel:
  - se complementează bitul k

Probleme

- Se citesc 2 numere întregi x și k. Să se afișeze numărul obținut din x astfel:
  - se complementează bitul k

$$x \;=\; a_1 \ldots\, a_{n-k}\; a_{n-(k-1)}\; a_{n-(k-2)} \;\ldots\; a_n$$

$$m \;=\; 0. \ldots. 0 \quad 1 \quad 0\ldots\ldots. 0$$

$$\color{red}{\mathbf{x \wedge m}} = a_1 \ldots\, a_{n-k}\; \mathbf{\sim a_{n-(k-1)}}\; a_{n-(k-2)} \;\ldots\; a_n$$

Probleme

- Se citesc 2 numere întregi x și k. Să se afișeze numărul obținut din x astfel:
  - se setează bitul numărul k la valoarea 1

    ```
    print(x | (1 << (k-1)))
    ```

  - se setează bitul k la valoarea 0

    ```
    print(x & (~ (1 << (k-1))))
    ```

  - se complementează bitul k

    ```
    print(x ^ (1 << (k-1)))
    ```

Probleme

- Să se verifice dacă un număr natural x este de forma 2^k

Probleme

- Să se verifice dacă un număr natural x este de forma 2^k

$$x = a_1 \ldots a_{k-1}\ 1\ 0\ 0 \ldots 0_{(2)}$$

Probleme

- Să se verifice dacă un număr natural x este de forma 2^k

$$x = a_1 \dots a_{k-1}\ 1\ 0\ 0 \dots 0_{(2)}$$

$$x - 1 =$$

Probleme

- Să se verifice dacă un număr natural x este de forma 2^k

$$x \quad = \quad a_1 \ldots a_{k-1}\ 1\ 0\ 0\ \ldots\ 0_{(2)}$$

$$x - 1 \quad = \quad a_1 \ldots a_{k-1}\ 0\ 1\ 1\ \ldots\ 1_{(2)}$$

$x \ \&\ (x - 1) =$

Probleme

- Să se verifice dacă un număr natural x este de forma 2^k

$$x = a_1 \dots a_{k-1}\ 1\ 0\ 0 \dots 0_{(2)}$$

$$x - 1 = a_1 \dots a_{k-1}\ 0\ 1\ 1 \dots 1_{(2)}$$

$$x\ \&\ (x - 1) = a_1 \dots a_{k-1}\ 0\ 0\ 0 \dots 0_{(2)}$$

Probleme

- Să se verifice dacă un număr natural x este de forma 2^k

$$x \quad = \quad a_1 \ldots a_{k-1}\ 1\ 0\ 0 \ldots 0_{(2)}$$

$$x - 1 \quad = \quad a_1 \ldots a_{k-1}\ 0\ 1\ 1 \ldots 1_{(2)}$$

$$x \ \& \ (x - 1) = a_1 \ldots a_{k-1}\ 0\ 0\ 0 \ldots 0_{(2)}$$

$$x \ \& \ (x - 1) = 0 \ \Leftrightarrow \ x = 1\ 0\ 0 \ldots 0_{(2)} \Leftrightarrow x \text{ este putere a lui } 2$$

Probleme

- Să se verifice dacă un număr natural x este de forma 2^k

```
print( x&(x-1) == 0)
```

Operatorul xor

- asociativ, comutativ
- $a \wedge 0 = a$ pentru $a \in \{0, 1\}$
- $a \wedge 1 = \sim a$ pentru $a \in \{0, 1\}$
- $a \wedge b = 1 \Leftrightarrow a \neq b$ pentru $a, b \in \{0, 1\}$
- $x \wedge x = 0$

Operatorul xor

- Să se interschimbe conținutul a două variabile x și y folosind xor

Operatorul xor

- Să se interschimbe conținutul a două variabile x și y folosind xor:

Avem

```
(x ^ y) ^ y = x
(x ^ y) ^ x = y
```

Operatorul xor

- Să se interschimbe conținutul a două variabile x și y folosind xor:

  Avem

  `(x ^ y) ^ y = x`

  `(x ^ y) ^ x = y`

  Notam **`aux = x ^ y`**

  Atunci intershimbarea se poate face astfel:

  ```
  y = aux ^ y
  x = aux ^ x
  ```

Operatorul xor

- Să se interschimbe conținutul a două variabile x și y folosind xor:

Avem

```
aux = x ^ y
y = aux ^ y ⇒ y devine x
x = aux ^ x
```

Putem renunța la aux?

Operatorul xor

- Să se interschimbe conținutul a două variabile x și y folosind xor:

Avem

```
aux = x ^ y
y = aux ^ y ⇒ y devine x
x = aux ^ x
```

Putem renunța la aux? – Da, il folosim pe x

```
x = x ^ y
y = x ^ y ⇒ y devine vechiul x
x = x ^ y
```

y este vechiul x

Operatorul xor

- Să se interschimbe conținutul a două variabile x și y folosind xor:

Avem

```
aux = x ^ y
y = aux ^ y ⇒ y devine x
x = aux ^ x
```

Putem renunța la aux? – Da, il folosim pe x

```
x = x ^ y
y = x ^ y
x = x ^ y
```

y este vechiul x

Exemplu

$x = 75, y = 20$

$x = x \wedge y = 75 \wedge 20$

$y = x \wedge y = (75 \wedge 20) \wedge 20 = 75$

$x = x \wedge y = (75 \wedge 20) \wedge 75 = 20$

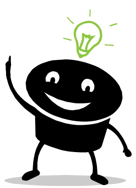
y este egal cu valoarea initiala a lui x

Probleme

- Se dau două numere naturale x și y. Calculați numărul biților din reprezentarea binară internă a numărului x a căror valoare trebuie comutată pentru a obține numărul y.

Probleme

- Se dau două numere naturale x și y. Calculați numărul biților din reprezentarea binară internă a numărului x a căror valoare trebuie comutată pentru a obține numărul y.



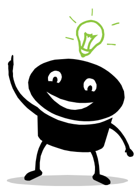
Se calculează numărul biților nenuli din x ^ y – implementare la laborator

Probleme

- Se citește un șir format din numere naturale cu proprietatea că fiecare valoare distinctă apare de exact două ori în șir, mai puțin una care apare o singură dată. Să se afișeze valoarea care apare o singură dată în șir.

Probleme

- Se citește un șir format din numere naturale cu proprietatea că fiecare valoare distinctă apare de exact două ori în șir, mai puțin una care apare o singură dată. Să se afișeze valoarea care apare o singură dată în șir.

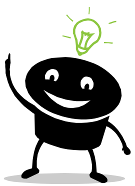


Fie $x_1, \ldots , x_n$ numerele din șir

Fie $v = x_1 \wedge \ldots \wedge x_n$

Probleme

- Se citește un șir format din numere naturale cu proprietatea că fiecare valoare distinctă apare de exact două ori în șir, mai puțin una care apare o singură dată. Să se afișeze valoarea care apare o singură dată în șir.



Fie $x_1, \dots , x_n$ numerele din șir

Fie $v = x_1 \wedge \dots \wedge x_n$

Deoarece $\wedge$ este asociativ și comutativ și $x \wedge x = 0$, xor pentru valorile care apar de 2 ori este 0, deci v este egal cu numărul care apare o singură dată

Probleme

- Să se calculeze numărul obținut prin aplicarea operatorului XOR între toate elementele tuturor submulțimilor unei mulțimi nevide de numere naturale $A=\{a_1,\ldots,a_n\}$ mai puțin mulțimea vidă. De exemplu, pentru mulțimea A $=\{2,7,4\}$ trebuie afișată valoarea

```
v = (2)^(7)^(4)^(2^7)^(2^4)^(7^4)^(2^7^4) = 0
```

Probleme

- Să se calculeze numărul obținut prin aplicarea operatorului XOR între toate elementele tuturor submulțimilor unei mulțimi nevide de numere naturale $A=\{a_1,\ldots,a_n\}$ mai puțin mulțimea vidă. De exemplu, pentru mulțimea $A=\{2,7,4\}$ trebuie afișată valoarea

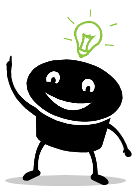
```
v = (2)^(7)^(4)^(2^7)^(2^4)^(7^4)^(2^7^4) = 0
```

- **De câte ori apare o valoare în v?**

Probleme

- Să se calculeze numărul obținut prin aplicarea operatorului XOR între toate elementele tuturor submulțimilor unei mulțimi nevide de numere naturale $A=\{a_1,\ldots,a_n\}$ mai puțin mulțimea vidă. De exemplu, pentru mulțimea $A=\{2,7,4\}$ trebuie afișată valoarea

```
v = (2)^(7)^(4)^(2^7)^(2^4)^(7^4)^(2^7^4) = 0
```



- De 2^{n-1} ori => pentru n> 1 apare de un număr par de ori => v=0
- Pentru n = 1 ??

Probleme – laborator

- Scrieți un program care testează dacă un număr natural n citit de la tastatură este de forma 2^k și, în caz afirmativ, afișează k

- Scrieți un program care determină numărul de biți egali cu 1 din reprezentarea binară a unui număr natural n citit de la tastatură