CURS 02 – FP

5. operatorii pe biți (doar pentru numere întregi):

~ (negare / not), & (și / and), | (sau / or), ^ (sau exclusiv / xor),
<< (deplasare la stânga / left shift), >> (deplasare la dreapta / right shift)

Reprezentarea numerelor întregi cu semn în complement față de 2

Numere pozitive	Numere negative
unsigned int x; //32 biți x = 23 = 00..010111 27 de biți 23 : 2 = 11, rest 1 11 : 2 = 5, rest 1 5 : 2 = 2, rest 1 2 : 2 = 1, rest 0 1 : 2 = 0, rest 1 $23_{(10)} = 10111_{(2)}$ = resturile de jos în sus	x = -24 |x| = 24 = 00..011000 ~|x| = ~24 = 11..100111 ~|x|+1 = ~24+1 = 11..101000 x = -24 = 11..101000

~ = negare / not (complement față de 1)

~	0	1	x = 23 = 00..010111
	1	0	~x = -24 = 11..101000

~x = -(x + 1) = -x -1, dacă x este un întreg cu semn

~x = ($2^{8*sizeof(x)}$ – 1) – x, dacă x este un întreg fără semn

```
#include<stdio.h>

int main()
{
    unsigned char x = 100;
    printf("%u", (unsigned char)~x);

    return 0;
}

//         100  = 01100100
//        ~100  = 10011011
// 100 + (~100) = 11111111 = 255 =>
//~100 = 255 - 100 = 155
```

Considerăm variabilele x și y de tip unsigned int!

& = și / and

&	0	1
0	0	0
1	0	1

```
      x = 349 = 0...000101011101
      y = 2006 = 0...011111010110
x & y = 340 = 0...000101010100
```

a & b = 1 <=> a = b = 1

| = sau / or

|	0	1
0	0	1
1	1	1

```
      x = 349 = 0...000101011101
      y = 2006 = 0...011111010110
x | y = 2015 = 0...011111011111
```

a | b = 0 <=> a = b = 0

^ = sau exclusiv / xor

^	0	1
0	0	1
1	1	0

```
      x = 349 = 0...000101011101
      y = 2006 = 0...011111010110
x ^ y = 1675 = 0...011010001011
```

a ^ b = 1 <=> a ≠ b

Criptare de tip Vernam:

EXPEDITORUL:

Mesaj clar = 1011000101011101

Cheia secretă (comună) = 0001011111010110

Mesaj criptat = Mesaj clar ^ cheia secretă = 1010011010001011

DESTINATARUL:

Mesaj criptat = 1010011010001011

Cheia secretă (comună) = 0001011111010110

Mesaj decriptat = Mesaj criptat ^ cheia secretă = 1011000101011101

CRIPTAT = CLAR ^ B

DECRIPTAT = CRIPTAT ^ B = CLAR ^ B ^ B = CLAR ^ 0 = CLAR

>> = deplasare spre dreapta / right shift
<< = deplasare spre stânga / left shift

unsigned int y = 2006 = 0...00111111010110

y >> 3 = 0000...00111111010~~110~~ = 250 = 2006 / 2^3

y = y >> b <=> y = y / 2^b

123456 >> 3 = 123 = 123456 / 10^3

Exemplu:

```
#include<stdio.h>

int main()
{
    unsigned int y = 2006;

    y = y >> 3;

    printf("y = %u", y);

    return 0;
}
```

y = 2006 = 11111010110

y << 3 = ~~000~~0...0111111010110000 = 16048 = 2006 * 2^3

y = y << b <=> y = y * 2^b (dacă nu apar supradepășiri!)

123456 << 3 = 123456000 = 123456 * 10^3

Exemple:

1. Testarea parității unui număr întreg

$$x = 349 = 0...000101011101$$

$$1 = 0...000000000001$$

$$x \,\&\, 1 = 0...000000000001$$

$$349 = 00...000101011101 = \underbrace{1*2^0}_{\text{0 sau 1}} + \underbrace{0*2^1 + 1*2^2 + ... + 0*2^{31}}_{\text{Valoare pară (sumă de puteri ale lui 2)}}$$

$$x \,\&\, 1 = \begin{cases} 1, & \text{dacă bitul de paritate al lui } x \text{ este } 1 \iff x \text{ este impar} \\ 0, & \text{dacă bitul de paritate al lui } x \text{ este } 0 \iff x \text{ este par} \end{cases}$$

2. Interschimbarea valorilor a două variabile de tip întreg (fără variabilă auxiliară)

x = 7 și y = 13 => x = 13 și y = 7

GREȘIT:

```
x = y;    // x = 13
y = x;    // y = 13
```

CORECT (cu variabilă auxiliară):

```
aux = x;  // aux = 7
x = y;    // x = 13
y = aux; // y = 7
```

CORECT (fără variabilă auxiliară):

```
x = x + y;     // x = x + y
y = x - y;     // y = (x + y) - y = x
x = x - y;     // x = (x + y) - x = y
```

Forma "prescurtată":

```
x = x + y - (y = x);
```

Proprietățile operatorului XOR:

a) t ^ t = 0
b) t ^ 0 = t
c) t ^ v = v ^ t
d) (t ^ v) ^ w = t ^ (v ^ w)

x ^ y = "diferențele" dintre x și y (din punct de vedere al biților aflați pe aceleași poziții)

```
x = x ^ y;
y = x ^ y;     // y = (x ^ y) ^ y = x ^ (y ^ y) = x ^ 0 = x
x = x ^ y;     // x = (x ^ y) ^ x = (x ^ x) ^ y = 0 ^ y = y
```

Forma "prescurtată":

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

Prioritățile operatorilor (vezi PDF)

```
x = 1, y = 5, z = 3
t = x + y * 2 => t = 1 + 10 = 11   //contează prioritățile
t = x + y + z => t = (x + y) + z = 6 + 3 = 9 //contează asociativitatea
```

Cazuri "ciudate":

```
1. x = 2 + 3 << 4;
```

GREȘIT: x = 2 + (3 << 4) = 2 + $3*2^4$ = 50
CORECT: x = (2 + 3) << 4 = $5*2^4$ = 80

2. x & 1 == 0 <=> x & (1 == 0) <=> x & 0 <=> 0

```
CORECT:
int x = 60;

if((x & 1) == 0)
     printf("Numarul %d este par!\n", x);
else
     printf("Numarul %d este impar!\n", x);
```

x & 1 == 1 <=> x & (1 == 1) <=> x & 1 <=> bitul de paritate

```
if(x)...  <=> if(x != 0)...
if(!x)... <=> if(x == 0)...
```

Funcții de citire/scriere

1. scriere -> funcția printf("formatări", expresii)

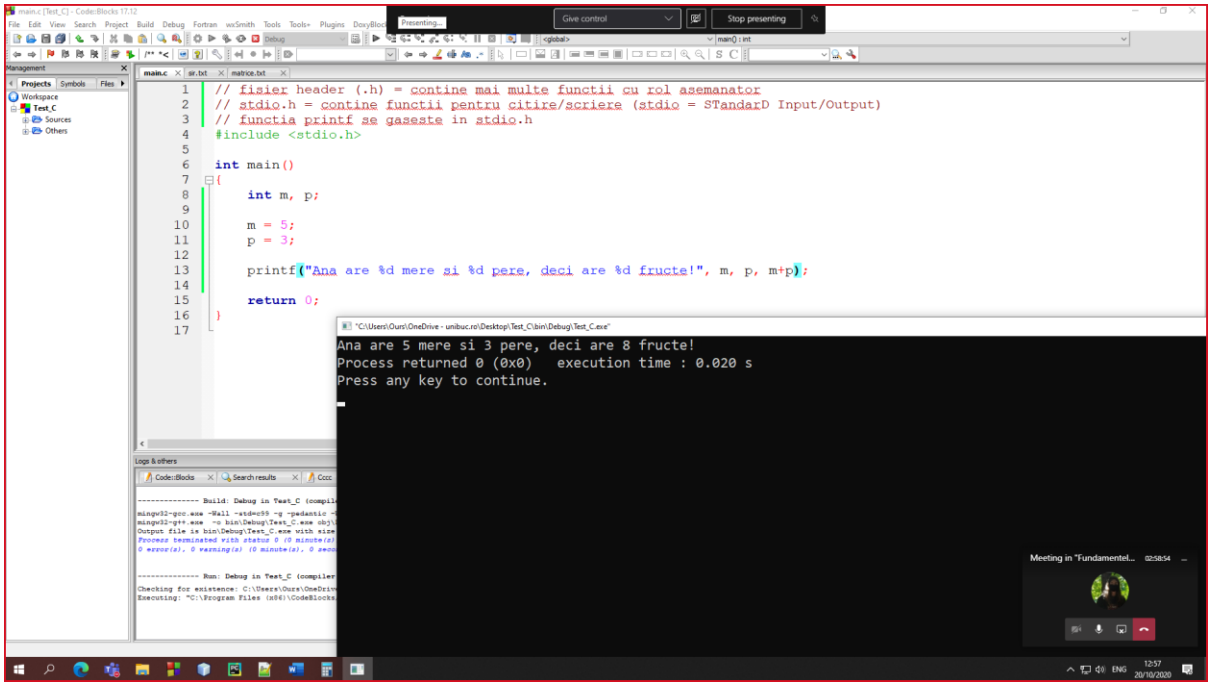
printf("Ana are mere!") -> "Ana are mere!" este un mesaj => se afișează neschimbat pe ecran

int m = 5, p = 3;
printf("Ana are %d mere!", m);
%d = specificator de format pentru numere întregi cu semn

printf("Ana are %d mere si %d pere!", m, p);

```
int m = 5, p = 3;

printf("Ana are %d mere si %d pere!", m, p);
```



2. citire -> funcția scanf("formatări", &variabilă_1, &variabilă_2,...)

```
#include<stdio.h>

int main()
{
    int x;
    unsigned int y;
    double t;

    printf("Introduceti cele 3 valori:\n");
    scanf("%d %u %lf", &x, &y, &t);

    printf("x = %d\n", x);
    printf("y = %u\n", y);
    printf("t = %f\n", t);

    return 0;
}
```

```
// fisier header (.h) = contine mai multe functii cu rol
asemanator
// stdio.h = contine functii pentru citire/scriere (stdio =
STandarD Input/Output)
// functia printf se gaseste in stdio.h
#include <stdio.h>

int main()
{
    int m, p;

//    printf("Numarul de mere: ");
//    scanf("%d", &m);
//
//    printf("Numarul de pere: ");
//    scanf("%d", &p);

    printf("Numarul de mere si numarul de pere: ");
    scanf("%d %d", &m, &p);

//    \n = salt la o linie noua
    printf("\nAna are %d mere si %d pere, deci are %d fructe!\n",
m, p, m+p);

    return 0;
}
```

Instrucțiunile limbajului C

1. instrucțiunea de decizie/alternativă

if(*expresie_logică*) **instrucțiune;**	**if(*expresie_logică*)** **instrucțiune_1;** **else** **instrucțiune_2;**

Bloc de instrucțiuni:

```
{
     instrucțiune_1;
     instrucțiune_2;
     ……
     instrucțiune_n;
}
```

Exemplu: maximul dintre două numere întregi

```
#include <stdio.h>

int main()
{
    int a, b, max;

    printf("Primul numar: ");
    scanf("%d", &a);

    printf("Al doilea numar: ");
    scanf("%d", &b);

    if(a > b)
        max = a;
    else
        max = b;

    printf("\nMaximul dintre %d si %d este %d!\n", a, b, max);

    return 0;
}
```