CURS 05 – FP

Algoritmi numerici fundamentali

1. Citirea valorilor de tip long long int/unsigned long long int în Windows

Se vor testa înainte specificatorii %llu și %lld!!!

```
#include <stdio.h>

int main()
{
    long long int x;
    unsigned long long int y;

    //introduceti pentru x si y valori
    //cel putin egale cu 5000000000 (cinci miliarde)

    printf("x = ");
    scanf("%lld", &x);
    printf("x = %lld\n\n", x);

    printf("y = ");
    scanf("%llu", &y);
    printf("y = %llu\n\n", y);

    //daca nu functioneaza varianta anterioara,
    //atunci folositi specificatorii %I64d si %I64u

    printf("x = ");
    scanf("%I64d", &x);
    printf("x = %I64d\n\n", x);

    printf("y = ");
    scanf("%I64u", &y);
    printf("y = %I64u\n", y);

    return 0;
}
```

2. Algoritmi care operează asupra cifrelor unui număr natural

n % 10 = ultima cifră a numărului n
n / 10 = numărul obținut prin eliminarea ultimei cifre din numărul n

5178 : 10 = 517, rest 8

Problema 1:
Să se calculeze suma cifrelor unui număr natural.

n	sc
5178	0
517	0 + 8 = 8
51	8 + 7 = 15
5	15 + 1 = 16
0	16 + 5 = 21

```
#include <stdio.h>

int main()
{
    unsigned int n, sc, aux;

    printf("n = ");
    scanf("%u", &n);
    aux = n;

    sc = 0;
    while(n != 0)
    {
        sc = sc + n%10;
        n = n / 10;
    }

    n = aux;
    printf("Suma cifrelor lui %u este %u\n", n, sc);

    return 0;
}
```

Problema 2:
Să se afișeze câte cifre pare și câte cifre impare conține un număr natural.

Exemplu: n = 21345 => 2 cifre pare și 3 cifre impare

```
#include <stdio.h>

int main()
{
    //nrp = numarul cifrelor pare
    //nri = numarul cifrelor impare
    unsigned long long int n, nrp = 0, nri = 0;

    printf("n = ");
    scanf("%llu", &n);

    if(n == 0) nrp=1;
    else
    {
        while(n!=0)
        {
            if(n%10%2 == 0)
                nrp++;
            else
                nri++;
            n= n/10;
        }
    }

    printf("Numarul de cifre pare: %llu\n", nrp);
    printf("Numarul de cifre impare: %llu\n", nri);

    return 0;
}
```

Problema 2:
Să se afișeze *cifra de control* a unui număr natural citit de la tastatură. *Cifra de control* a unui număr natural se obține calculând, în mod repetat, suma cifrelor sumei cifrelor numărului dat.

Exemplu:
n = 987569978 => sc = 68 => sc = 14 => sc = 5

n = 987569978 => sc = 68
n = 68 => sc = 14
n = 14 => sc = 5 <= 9

```c
#include <stdio.h>

int main()
{
    unsigned long long int n, cp;
    unsigned int sn;

    printf("n = ");
    scanf("%llu", &n);

    cp = n;

    while(n >= 10)
    {
        sn = 0;
        while(n != 0)
        {
            sn = sn + n%10;
            n = n/10;
        }
        n = sn;
    }

    printf("Cifra de control a numarului %llu este %u", cp, n);

    return 0;
}
```

Variantă (fără instrucțiuni repetitive!)

$$n = \overline{c_{k-1}c_{k-2}\ldots c_1c_0} = c_{k-1}\times 10^{k-1} + c_{k-2}\times 10^{k-2} + \cdots + c_1\times 10^1 + c_0$$

$$sn = c_{k-1} + c_{k-2} + \cdots + c_1 + c_0$$

$$n - sn = (c_{k-1}\times 10^{k-1} - c_{k-1}) + (c_{k-2}\times 10^{k-2} - c_{k-2}) + \cdots + (c_1\times 10^1 - c_1) + (c_0 - c_0)$$

$$n - sn = c_{k-1}(10^{k-1} - 1) + c_{k-2}(10^{k-2} - 1) + \cdots + c_1(10^1 - 1)$$

$$n - sn = c_{k-1}\times \underbrace{99\ldots9}_{k-1\text{ ori}} + c_{k-2}\times \underbrace{99\ldots9}_{k-2\text{ ori}} + \cdots + c_1\times \underbrace{9}_{1\text{ ori}}$$

$$(n - sn) \vdots 9 \Leftrightarrow n \,\%\, 9 = sn \,\%\, 9$$

Teoremă:
Restul împărțirii unui număr natural la 9 este egal cu restul împărțirii sumei cifrelor numărului respectiv la 9.

Cazul 0: $n = 0 => cc = 0$
Cazul 1: $n \,\%\, 9 = 0 => sn \,\%\, 9 = 0 => sn = 9 => cc = 9$
Cazul 2: $n \,\%\, 9 = r \neq 0 => sn \,\%\, 9 = r => cc = r$

```c
#include <stdio.h>

int main()
{
    unsigned long long int n;

    printf("n = ");
    scanf("%llu", &n);

    if(n % 9 == 0)
        printf("Cifra de control a numarului %llu este 9", n);
    else
        printf("Cifra de control a numarului %llu este %llu", n, n % 9);

    return 0;
}
```

Problema 3:
Așteptându-și partenerii de rummy, Gigel se joacă cu p piese ($p \leq 15$) având valori cuprinse între 1 și 9 (indiferent de culoare) astfel: formează pe tablă un număr și apoi mută ultima piesă din dreapta pe prima poziție, în mod repetat, de exact p ori. Plictisit, Gigel se gândește cum ar putea afla suma tuturor numerelor pe care le-a format în acest mod, știind doar numărul inițial (care, de fapt, se află chiar acum pe tablă). Voi puteți să scrieți un program care să-l ajute pe Gigel?

Exemplu:

n =		2	7	3	8	5	1	+
		1	2	7	3	8	5	
		5	1	2	7	3	8	
		8	5	1	2	7	3	
		3	8	5	1	2	7	
		7	3	8	5	1	2	
		~~2~~	~~7~~	~~3~~	~~8~~	~~5~~	~~1~~	
		26	26	26	26	26	26	
s =	2	8	8	8	8	8	6	

Observații:

- Jocul de Rummy conține **106 piese**: 104 piese cu numere și 2 piese de **Jolly**. Piesele cu numere sunt împărțite în **4 culori:** roșu, galben, albastru și negru. Fiecare culoare conține numerele de la 1 la 13 în dublu exemplar.
- Se revine la numărul inițial după un număr de pași egal cu numărul de cifre ale lui n.

Varianta 1: forță brută = se calculează fiecare număr și se adună la o sumă

```
nr_cifre = 6
2 7 3 8 5 1 : 10 = 2 7 3 8 5, rest 1
1 2 7 3 8 5 = 1 * 10^5 + 2 7 3 8 5
```

```
#include <stdio.h>

int main()
{
    unsigned long long int n, aux;

    //nr_cifre = numarul de cifre ale lui n
    unsigned char i, nr_cifre;

    //p = 10 la puterea (nr_cifre - 1)
    //s = suma ceruta
    unsigned long long int p, s;

    printf("n = ");
    scanf("%I64u", &n);

    aux = n;

    p = 1;
    nr_cifre = 0;
    while(n != 0)
    {
        p = p * 10;
        nr_cifre++;
        n = n / 10;
    }
    p = p / 10;
```

```
    n = aux;
    s = 0;
    for(i = 0; i < nr_cifre; i++)
    {
        s = s + n;
        n = n % 10 * p + n / 10;
    }

    printf("\nSuma: %I64u\n", s);

    return 0;
}
```

Observație:

Numărul de cifre p ale unui număr natural nenul n este $p = 1 + [\log_{10} n]$.

$$4 \leq \log_{10} 12345 < 5 => \ p = 1 + [\log_{10} 12345] = 5$$

Presupunem că numărul n are p cifre => $10^{p-1} \leq n \leq 10^p - 1 =>$
$=> 10^{p-1} \leq n < 10^p => \log_{10} 10^{p-1} \leq \log_{10} n < \log_{10} 10^p =>$
$=> p - 1 \leq \log_{10} n < p => [\log_{10} n] = p - 1 => p = 1 + [\log_{10} n]$

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

int main()
{
    unsigned long long int n;

    //nr_cifre = numarul de cifre ale lui n
    unsigned int i, nr_cifre;

    //p = 10 la puterea (nr_cifre - 1)
    //s = suma ceruta
    unsigned long long int p, s;

    printf("n = ");
    scanf("%I64u", &n);

    nr_cifre = 1 + (unsigned int)log10(n);
    p = 1;
    for(i = 0; i < nr_cifre - 1; i++)
        p = p * 10;

    s = 0;
    for(i = 0; i < nr_cifre; i++)
    {
        s = s + n;
```

```
        n = n % 10 * p + n / 10;
    }

    printf("\nSuma: %I64u\n", s);

    return 0;
}
```

Varianta 2:

2 7 3 8 5 1 = $2*10^5$ + $7*10^4$ + ... + $5*10^1$ + $1*10^0$
1 2 7 3 8 5 = $1*10^5$ + $2*10^4$ + ... + $8*10^1$ + $5*10^0$
....
7 3 8 5 1 2 = $7*10^5$ + $3*10^4$ + ... + $1*10^1$ + $2*10^0$

s = (2 + 1 + ... + 7)$*10^5$ + ... + (1 + 5 + ... + 2)$*10^0$
= suma_cifrelor_lui_n * (10^5 + 10^4 + ... + 10^0) =
= suma_cifrelor_lui_n * 111...1
unde cifra 1 apare de un număr de ori egal cu numărul cifrelor lui n.

- Valoare maximă a sumei cerute se obține pentru 8 cifre egale cu 9 și 7 cifre egale cu 8, respectiv pentru n = 999.999.998.888.888, deci valoarea maximă a sumei cerute este cel mult 128 * 111.111.111.111.111 = 14.222.222.222.222.208 și este strict mai mică decât ULLONG_MAX = 18.446.744.073.709.551.615.

```
#include <stdio.h>

int main()
{
    unsigned long long int aux, n;

    //u = 111...1
    //sc = suma cifrelor lui n
    //s = suma ceruta
    unsigned long long int p, s, sc;

    printf("n = ");
    scanf("%I64u", &n);

    aux = n;

    sc = p = 0;
    while(n != 0)
    {
        sc = sc + n % 10;
        p = p * 10 + 1;
```

```
        n = n / 10;
    }

    s = sc * p;

    printf("\nSuma: %I64u\n", s);

    return 0;
}
```

3. Testarea primalității unui număr:

Un număr este *prim* dacă se divide doar cu 1 și el însuși.
Un număr este *prim* dacă nu are divizori proprii (diferiți de 1 și el însuși).

```
#include <stdio.h>

int main()
{
    int n, d;

    printf("n = ");
    scanf("%d", &n);

    for(d = 2; d <= n/2; d++)
        if(n % d == 0)
            break;

    if(d == n/2 + 1)
        printf("Numarul %d este prim!", n);
    else
        printf("Numarul %d este compus!", n);

    return 0;
}
```

Observație: Toți divizorii proprii ai unui număr natural n sunt cuprinși între 2 și $\left[\frac{n}{2}\right]$.
Exemplu: $n = 15 => \mathcal{D}_{15} = \{1, 3, 5\} =>$ toți divizorii sunt mai mici sau egali decât $[15/2] = 7$.

Teoremă: Dacă n este compus, atunci el are cel puțin un divizor propriu mai mic sau egal decât $\sqrt{n}$.
Exemplu: $n = 15 => d_1 = 3 < \sqrt{15} \approx 3.87$, dar $d_2 = 5 > \sqrt{15}$!

Demonstrație:

Pp. prin absurd faptul că n este compus, dar nu are niciun divizor propriu mai mic sau egal decât $\sqrt{n}$ => $n = d_1 * d_2$ și $d_1, d_2 > \sqrt{n}$ => $n = d_1 * d_2 > \sqrt{n} * \sqrt{n} = n$ (imposibil).

```
#include <stdio.h>

int main()
{
    int n, d;

    printf("n = ");
    scanf("%d", &n);

    //d * d <= n este echivalenta cu d <= sqrt(n),
    //dar mult mai rapida!!!
    for(d = 2; d * d <= n; d++)
        if(n % d == 0)
            break;

    if(d * d > n)
        printf("Numarul %d este prim!", n);
    else
        printf("Numarul %d este compus!", n);

    return 0;
}
```