b2021_04

Subiectul I

1 – d.

2.

$f(1234) = 0 * 4 + f(123) = 4$

$f(123) = 1 \cdot 3 + f(12) = 3 + 1 = 4$

$f(12) = 0 \cdot 2 + f(1) = 1$

$f(1) = 1 \cdot 1 + f(0) = 1$

// programul face suma cif. impare adică din 1234, 1+3 = 4

=> b

3. a) 4. c) 5. d b c) X ?

1 – 11, 2 – 12, 3 – 13, 4 – 14, 5 – 15,
6 – 16, 7 – 17, 8 – 18, 9 – 19, 10 – 20, 11 –
11 – 21, 1 – 21;

Subiectul II

1. a) 3_4_8_9_15_ b) 1, 1, 1 ; 1, 1, 2 ;

c)

```
#include <iostream>
using namespace std;
int main(){
    int n, x, y;
    cin>>n,
     cin >> n >> x >> y;
    bool ok = false;
    for (int i = 1; i <= n; i++) {
        if((i%x==0 && i%y!=0){
        || (i%x!=0 && i%y==0)) {
            cout << i << " ";
            ok = true;
        }
    }
    if (!ok) { cout << 0 <<;}
    Return 0;
}
```

2.

and	0	1	2	3	4	5	6	7	i	s
~~A~~	R	~~R~~	M	~~A~~	N	I	~~A~~	O	~~A-1~~	~~3~~
~~A~~		~~A~~		~~A~~			A		~~2~~	~~2~~
A		A		O					~~1~~	~~1~~
		A	R						0	0

⇒ S = "ARMONIA"

3. a[0].anNastere = 2000;

a[0].varsta = 4000;

Subiectul III 1.

```
Void cuburi (int n) {
    ~~int vec[1000]~~ int vec[1000];
    ~~for (int~~ i = n; i > 0; i++) {
    int q = 1;
    for (int i = 0; i < n; i++) {
        vec[i] = q * q * q;
    }
    for (int i = n; i > 0; i++) {
        cout << ~~vec[i]~~ vec[i-1]; << " ";
    }
}
```

2.

```
int include <iostream>
using namespace std;
int main () {
    int k; int n;
    cin >> k >> n;
    tab [1000][1000];
    for(int i =0; i<n; itt) {
        tab [i][i] = (i+1) · k;
    }
    for (int i = 0; i < n; itt){
        for (int j =0; j < n; jtt) {
            tab[i][j] = tab[j][j] - tab [j][j]
                tab[i][i]
                -(
            tab[i][j] - tab[i][i] +(j-i)
                                  -(i-j);
        }
    }
    for(int i =0; i<n*n;itt) {cout<<tab [i/n][i%n] << " ";
        if (n*0 i%n ==0 tt && n>0) {cout << "\n";}
    } return 0;
}
```

4/

3.

```
#include <iostream>
#include <fstream>

using namespace std;

bool is_armstrong(int x, int y){

    x = x % 100; y = y % 100; int inv = (x%10)*10 + x/10;

    if (x == y return (x == y || x == inv);
}
int fv_a[1000]; int fv_b[10000];
int main(){
    int na; int nb; fin >> na >> nb;

    for (int i = a; i < na; i++) { int x = 0;
    fin >> x; fv_a[x]++; }
    for (int j = 0; j < nb; j++) { int y = 0; fin >> y;
    fv_b[y]++; } int s = 0;

    for (int k = 10; k < 100; k++){
        int inv = (k % 10) * 10 + k/10;
        s = fv_a[k] * fv_b[k]; + fv[k] = fv_a[k] * fv_b[inv];
        s = fv_a[k] * fv_b[inv];
} cout

    s = (f   s = fv_a[k] * fv_b[k]
             + fv_a[k] * fv_b[inv]; }
cout << s; return 0; }
```

5/

Algoritmul funcționează pe baza faptului că, pentru proprietatea discutată, importante sunt doar ultimele 2 cifre ale oricărui număr citit. Programul de aici citește toate numerele dispuse în fișier și memorează doar frecvența fiecărei secvențe de 2 cifre de la finalul numărului (cu ajutorul unui vector cu 90 de poziții importante), ~~o dată~~ de câte ori, o dată pentru fiecare șir de cuvinte din fișier. Luând în considerare modul de alegere al perechilor programul apoi iterează peste vectorii de frecvență și înmulțește perechile prin bara produsului sufixului extras din primul vector (fv_a) cu sufixul din cel de al doilea (fv_b), cât și cu inversul lui din fv_b. ~~Programul deci înmulțește doar~~ În final programul afișează cantitatea de perechi dorită, stocată în registrul S.

Programul este eficient din punct de vedere al timpului de execuție deoarece parcurge minimul necesar de instrucțiuni pentru a citi valorile fișierului (na + nb), făcând apoi doar un număr constant de 100 de pași pentru a îndeplini cerința. Gradul de complexitate al programului este așadar O(n) unde n = (na + nb).

6/