Rezolvări și explicații – set 1

1. Să se genereze şirul lui Fibonacci pentru primele “N” numere.
   Rezolvare:

```c
#include <stdio.h>
#include <stdlib.h>

int main()
{
    int n, i, nr1=0, nr2=1, nr3;
    printf("Cate numere Fibonacci doriti sa generati?\n");
    printf("n="); scanf("%d",&n);
    printf("Sirul Fibonacci pentru primele %d numere este: ", n);
    printf("%d %d", nr1, nr2);
    i=2; // contorul "i" numara cate numere Fibonacci am tiparit
    // la un moment dat de timp in consola programului
    while (i<n){
        nr3=nr1+nr2;//numarul al 3-lea va fi suma celor 2 numere
        // generate anterior
        printf(" %d", nr3);
        i++; // incrementam "i" cu o unitate fiindca am tiparit
        // un nou numar din sirul Fibonacci
        nr1=nr2; //reactualizam primul numar
        nr2=nr3; //reactualizam si al 2-lea numar
    }
    return 0;
}
```

2. Se dau 2 numere naturale A şi B. Să se afle cel mai mare divizor comun.

Rezolvare:

Metoda 1

```c
#include <stdio.h>
#include <stdlib.h>

int main()
{
    int a,b,d,s,r;
    printf("Numarul a=");scanf("%d",&a);
    printf("Numarul b=");scanf("%d",&b);
    d=a; //d - descazut
    s=b; //s - scazator
    while (d!=s){
        r=abs(d-s); //r - rest
        // abs() este functia modul din libraria stdlib.h
        d=s;
        s=r;
    };
    printf("Cel mai mare divizor comun este %d\n",d);
    return 0;
}
```

Metoda 2 – prin scăderi repetate

```c
#include <stdio.h>
#include <stdlib.h>

int main()
{
    int a,b;
    printf("Numarul a=");scanf("%d",&a);
    printf("Numarul b=");scanf("%d",&b);
    while (a!=b){
        if (a>b) a=a-b;
        else b=b-a;
    };
    printf("Cel mai mare divizor comun este %d\n",a);
    return 0;
}
```

Metoda 3 – prin împărțiri repetate

```c
#include <stdio.h>
#include <stdlib.h>

int main()
{
    int a,b,r; // r - rest
    printf("Numarul a=");scanf("%d",&a);
    printf("Numarul b=");scanf("%d",&b);
    while (r=a%b) // cat timp r diferit de zero
    // unde prin atribuire, r este a%b
    // "r" -stocheaza restul imparitii lui "a" la "b"
    {
        a=b;
        b=r;
    }
    printf("Cel mai mare divizor comun este %d\n",b);
    return 0;
}
```

3. Se dă un număr natural "N". Să se testeze dacă este prim sau nu.

Rezolvare:

Metoda 1

```c
#include <stdio.h>
#include <stdlib.h>

int main()
{
    // in limbajul C nu exista tipul boolean
    // ca sa stochez valoarea de adevar (true sau false) a unei propozitii
    // atunci pentru encodare se foloseste 1 pentru true
    // si 0 pentru false
    // initial presupun ca numarul "n" citit este prim
    // valoarea de adevar a acestei supozitii o stochez in variabila prim
    // deci prin prim=1 presupun ca "n" este prim
    // "d" este divizorul; initial 2
    int n, prim=1, d=2;
    printf("n="); scanf("%d", &n);
    // ca sa testez daca un numar "n" este prim
    // merg cu divizorii de la 2 pana la jumatatea numarului
    // deci "d" apartine intervalului [2;n/2] de valori
    do
    {
        // daca numarul se imparte in mod exact la divizor
        // altfel spus, daca restul impartirii lui n la d e 0
        // Obs: in expresii % este operator pentru rest
        // conditia a 2-a si anume (n!=2) am pus-o pt optimizare
        // ca programul sa functioneze si pt n=2
        if ((n%d==0) && (n!=2)){
            prim=0; // ceea ce am presupun devine fals
            break; // intrerup executia
        }
        d++; // incrementez "d" ca sa verific conditia si pt ceilalti divizori
    }while (d<(n/2) && (prim==1)); // cat timp mai am valori in interval
    // si ceea ce am presupus este adevarat (prim==1) ca "n" e prim
    // && este "si"-ul pentru conditii compuse
    if (prim==0)
        printf("%d NU este numar prim\n",n);
    else
        printf("%d este numar prim\n",n);
    return 0;
}
```

Observație 1: O altă alternativă de rezolvare este să merg cu divizorii de la 2 până la $\sqrt{n}$, deci divizorul $d \in [2\,;\sqrt{n}]$ iar atunci condiția din bucla cu test final este:

```c
do{

}while (d<(sqrt(n)) && (prim==1));
```

Observație 2: Algoritmul de mai sus poate folosi o buclă cu test inițial pentru a verifica dacă un număr e prim sau nu, iar echivalentul buclei cu test final în acest caz este:

BUCLĂ CU TEST FINAL	BUCLĂ CU TEST INIȚIAL

`do`
`{`
`  if ((n%d==0) && (n!=2)){`
`  prim=0;`
`  break;`
`  }`
`  d++;`
`}while (d<(n/2) && (prim==1));`
	`while (d<=(n/2) && (prim==1)){`
	`  if ((n%d==0) && (n!=2)){`
	`  prim=0;`
	`  break;`
	`  }`
	`  d++;`
	`}`

Metoda 2

```c
#include <stdio.h>
#include <stdlib.h>

int main()
{
    int n, prim=1,d;
    printf("n="); scanf("%d", &n);
    for(d=2;d<=(n/2);d++) // alternativa for(d=2;d<=sqrt(n);d++)
    {
        if (n%d==0)
            prim=0;
    }
    if (prim==0)
        printf("%d NU este numar prim\n",n);
    else
        printf("%d este numar prim\n",n);
    return 0;
}
```

4. Se dă un număr natural “N”. Să se descompună în factori primi.

Rezolvare:

```c
#include <stdio.h>
 #include <stdlib.h>

 int main()
 {
     unsigned int n,d,p;
     printf("n=");scanf("%d",&n);
     d=2; //primul factor prim
     printf("%d = ", n);
     while(n>1){
         p=0; // "p" stocheaza nr de repetari a unui factor prim (puterea)
         while (n%d==0){
             p++;
             n=n/d;
         };
         if (p>0){
             if (n==1) printf("(%d^%d)",d,p);
             else printf("(%d^%d) * ",d,p);// "n" se divide de "p" ori cu "d"
```

```
19              }
20              d++;
21          }
22          return 0;
23      }
```