1.Pentru valori întregi citite de la tastatură să se tiparească valoarea corespunzătoare în binar.

Ne luam o masca folosind numarul 1 in binar si shiftandul cu i pozitii. Astfel, cand facem SI binar cu numarul N, vom obtine 0 pe toate pozitiile (daca bitul i este 0 in n), fie 2^n (daca n are bitul i cu valoarea 1). Nu putem afisa direct variabila bit, pentru ca ar afisa 2^n pentru cazul cu 1. Avem nevoie sa punem bitul inapoi pe pozitia 0, shiftand iarasi cu i pozitii la dreapta.

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

int main()
   {
        int n,i=0;
        scanf("%d",&n);
        for(i=15;i>=0;--i)
        {
                int bit=n&(1<<i);
                int x=bit>>i;
                printf("%d",x);
        }
        scanf("%d",&n);
        return 0;
   }
```

2.Se citesc 2 numere întregi x şi n unde n este între 0 si 15. Să se afişeze: bitul n din x, numărul x în care se seteaza bitul n, numărul x în care se şterge bitul n, numărul x în care se complementează bitul n.

Pentru afisare, shiftam pe x cu n pozitii la dreapta, astfel inca bitul n sa ajunga pe ultima pozitie. Apoi putem face SI binar cu 1.

Pentru stergere, avem nevoie de o masca ce are 1 pe toate pozitiile si 0 pe pozitia n (daca facem si cu aceasta masca, totul va ramane la final inafara de pozitia n, care va ajunge 0). Facem asta prin shiftarea lui 1 cu n pozitii si negarea acestuia.

Setarea este similara cu stergerea. Operatia SAU binar ne ofera procedeul de setare (daca bitul n este 0 sau 1 in x, cu 1 din masca, operatia va da 1).

Complementarea se foloseste de XOR binar, pentru ca acesta ne da cazurile 0 1 -> 1 si 1 1 -> 0. Al doilea bit este cel setat de noi din masca, iar primul este bitul n din x.

```
#include <stdio.h>

int main()
{
   int x, n;
   scanf("%d%d",&x,&n);

   printf("Afisare bit n: %d\n",(x>>n)&1);
   printf("Stergere: %d\n", x&~(1<<n));
   printf("Setare: %d\n", x|1<<n);
   printf("Complement: %d\n",x^(1<<n));
```

```
    scanf("%d",&n);
    return 0;
}
```

3. Se citesc întregii x, y, n, p. Să se copieze în x, începând din poziţia p, ultimii n biţi din y şi să se afişeze noua valoare a lui x.

Cum v-am spus, daca facem modulo 2^n, obtinem ultimii n biti dintr-un numar. Puteti testa pe hartie. De exemplu: 35 = 0010 0011. Vrem ultimele 3 cifre. 35%(2^3)=35%8=3(11 sau 011 in binar).

Daca vreti o solutie ce foloseste operatori binari in totalitate, un coleg de la grupa de joi, cred ca Teodorescu Stefan, a propus o solutie ce implica shiftarea la dreapta cu 32-n biti, pe cazul general sizeof(variabilaInCareStocamNumarul), si apoi shiftarea la dreapta cu tot atatia biti. Foarte buna solutie :D.

O alta solutie buna a venit de la un coleg de vineri, din nou, imi cer scuze ca nu v-am invatat numele deloc. El a propus sa facem o masca de biti (cum am tot facut la laboratorul la exercitiul 2, mastile acelea ce aveau 0 pe toate pozitiile mai putin pozitia N, sau cele aveau numai 1 si 0 pe pozitia N). Ne construim o masca ce are 1 pe ultimele N pozitii (sunt mai multe metode a face asta, de exemplu, setand fiecare bit pe rand pe 1, ca in exercitiul 2) si facem un SI binar cu numarul dat. Foarte ingenios, de asemenea :D.

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

int main()
{
        int x,y,n,p;
        scanf("%d %d %d %d", &x,&y,&n,&p);

        x=x%(int)pow(2,p);
        y=y%(int)pow(2,n);
        y=y<<p;
        x=x|y;

        printf("\n%d\n\n",y);
        return 0;
}
```

4. Scrieţi un program care primeşte ca input de la tastatură scrierea unui număr în baza 2 şi calculează direct scrierea acestuia în baza 16 (nu mai trece prin baza intermediară 10). Realizaţi acest lucru inversând cele două baze (input – scrierea în baza 16, output – scrierea în baza 2).

Ne facem un vector cu toate valorile posibile in hexa. Citim un numar binar drept long long int (e un exercitiu bun sa-l facem si ca string, dar am ales calea mai usoara). Cat timp numarul binar este 0, ii luam ultimele 4 cifre, facand mod 10000. Apoi testam aceste 4 cifre impotriva vectorului ales mai devreme. Daca cele 4 cifre reprezinta un simbol hexa intre 0 si 9, afisam i + 48 (de la 48 incep cifrele in tabelul ASCII http://www.asciitable.com/). Daca trebuie sa scriem A B C D E sau F, atunci incepem de la 65 plus (i-10) pentru ca A=65, B =66 etc. Intre timp, trebuie sa tinem o variabila index ca sa stim unde in vectorul de hex sa punem rezultatul. La finalul buclei while, ne asiguram ca elimnam ultimele 4 cifre din numar, impartind la 10000.

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

int main()
{
	int valoriHexa[] = {0, 1, 10, 11, 100, 101, 110, 111, 1000,
		1001, 1010, 1011, 1100, 1101, 1110, 1111};

	long long int binar;
	char hex[20];
	int index, i, cifra;

	printf("Numar in binar: ");
	scanf("%lld", &binar);

	index = 0;

	while(binar!=0)
	{
		/* Ultimele 4 cifre */
		cifra = binar % 10000;

		/* Facem match ultimelor 4 cifre cu vectorul de constante hexa */
		for(i=0; i<16; i++)
		{
			if(valoriHexa[i] == cifra)
			{
				if(i<10)
				{
					/* pentru constante 0-9 */
					hex[index] = (char)(i + 48);
				}
				else
				{
					/* pentru constante char A-F */
					hex[index] = (char)((i-10) + 65);
				}
```

```
                        index++;
                        break;
                    }
                }

                /* Taiem ultimele 4 cifre. */
                binar /= 10000;
            }
            hex[index] = '\0';

            /* Inversam ordinea numarului pentru ca noi am citit
            de la capatului numarului binar spre inceput. */
            strrev(hex);

            printf("Hexadecimal number = %s", hex);

            return 0;
}
```

Pentru hex spre binar, nu prea e mult de zis la partea asta. Doar citim un numar hexa intr-un string si facem o verificare pe fiecare caracter, mapand fiecare posibilitate la un caz in switch.

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

int main()
{
        char hexString[8];
        char binar[32];

        printf("Numar hexa:");
        scanf("%s",&hexString);

        int len=strlen(hexString);
        for(int i=0;i<len;++i)
        {
                switch(hexString[i])
                {
                case '0': printf("0000");break;
                case '1': printf("0001");break;
                case '2': printf("0010");break;
                case '3': printf("0011");break;
                case '4': printf("0100");break;
                case '5': printf("0101");break;
                case '6': printf("0110");break;
                case '7': printf("0111");break;
                case '8': printf("1000");break;
                case '9': printf("1001");break;
                case 'A': case 'a': printf("1010");break;
                case 'B': case 'b': printf("1011");break;
                case 'C': case 'c': printf("1100");break;
                case 'D': case 'd': printf("1101");break;
                case 'E': case 'e': printf("1110");break;
```

```
                case 'F': case 'f': printf("1111");break;
                default:
                        printf(" ");
                }
        }
        return 0;
}
```

5. Se citesc numere naturale până la întâlnirea numărului 0. Să se afişeze toate perechile de numere consecutive citite cu proprietatea că al doilea număr reprezintă restul împărţirii primului număr la suma cifrelor sale.

Nu sunt multe de zis despre problema asta. Cat timp userul nu ne da un 0, facem suma cifrelor si afisam perechea, daca conditia din cerinta este indeplinita. If-ul if(precedent != 0) ne ajuta sa avem prima pereche in regula si sa nu fie considerata valoarea de initializare a lui precedent drept numar introdus (adica userul trebuie sa introduca doua numere pentru ca if-ul ce verifica conditia problemei sa se ruleze).

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

int sumaCifre(int n)
{
        int sum=0;
        while(n)
        {
                sum+=n%10;
                n/=10;
        }
        return sum;
}


int main()
{
        int n=1,i=0, precedent=0;
        while(n != 0)
        {
                scanf("%d",&n);
                if(precedent != 0)
                {
                        int sum=sumaCifre(precedent);
                        if(n == (precedent%sum))
                        {
                                printf("Perechea: %d %d\n",precedent,n);
                                i++;
                        }
                }
                precedent=n;
        }
        scanf("%d",&n);
        return 0;
}
```

6. Se citeşte de la tastatură un număr natural p. Să se determine toate perechile distincte de numere întregi (i,j,k) cu proprietatea că ele pot reprezenta laturile unui triunghi de perimetru p. Folosiţi maxim două instrucţiuni for.

Cum v-am zis la laborator, vom afisa perechi (i, j, k) drept (i, j, p-i-j), iar frumusetea problemei sta in modul in care scriem cele doua for-uri. Primul for pleaca de la 1 la p-1 (de la 1 pentru ca nu vreau cazul cu i=0, pana la p-1 pentru ca nu vreau cazul j=0). Al doilea for pleaca cu j de la valoarea i, pentru ca altfel vom avea in duplicate in output si se opreste la p-i-1. Partea cu p-i pentru ca altfel i+j va fi mai mare decat p, si acel -1 pentru ca plecam de la 1 cu i.

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

int main()
{
   int p;
   scanf("%d",&p);
   for(int i=1;i<p-1;++i)
   {
         for(int j=i;j<p-i-1;++j)
         {
               printf("%d %d %d\n",i,j,p-i-j);
         }
   }
   scanf("%d",&p);
   return 0;
}
```