TP-Recursividad

Ejercicio 1
Implemente una función recursiva que nos diga si una cadena es palíndromo.

Ejercicio 2:
Escriba una definición recursiva de una función que tiene un parámetro n de tipo entero y que devuelve el n-ésimo número de Fibonacci. Los números de Fibonacci se definen de la siguiente manera:
$F_0 = 1$
$F_1 = 1$
$F_{i+2} = F_i + F_{i+1}$

Ejercicio 3:

- Dado el siguiente código y explicación, establecer la o las diferencias entre las mismas.
- En caso de diferencia, adecuar el código para que cumpla con el texto.

```c
#include <stdio.h>
int lcmCalculate(int a, int b);

int main()
{
    int n1, n2, lcmOf;
    printf("\n\n Recursion : Find the LCM of two numbers :\n");
    printf("-----------------------------------------------\n");
    printf(" Input 1st number for LCM : ");
    scanf("%d", &n1);
    printf(" Input 2nd number for LCM : ");
    scanf("%d", &n2);
// Ensures that first parameter of lcm must be smaller than 2nd
    if(n1 >  n2)
        lcmOf = lcmCalculate(n2, n1);//call the function lcmCalculate for lcm calculation
    else
        lcmOf = lcmCalculate(n1, n2);//call the function lcmCalculate for lcm calculation
    printf(" The LCM of %d and %d :  %d\n\n", n1, n2, lcmOf);
    return 0;
}
int lcmCalculate(int a, int b)//the value of n1 and n2 is passing through a and b
{
    static int m = 0;
    //Increments m by adding max value to it
    m += b;
//  If found a common multiple then return the m.
    if((m % a == 0) && (m % b == 0))
    {
        return m;
    }
    else
    {
        lcmCalculate(a, b);//calling the function lcmCalculate itself
    }
}
```

La función recursiva lcm() toma dos números enteros como argumentos. Tome una variable estática m e inicialícela con cero, luego actualícela con uno de los números. Aquí actualizamos la variable m con b, por lo que la variable m siempre será divisible por la variable b. Por lo tanto, necesitamos verificar solo con la variable a, es decir (m%a == 0).

Como estamos sumando el número más grande a la variable m, entonces la variable b debe contener el número más grande. Al llamar a lcm(), debemos pasar el número menor como a y el número mayor como b.

Si sumamos el número mayor, el número de llamadas recursivas será el mínimo, pero si sumamos el número menor, el número de llamadas recursivas será mayor que en el caso anterior. Sabemos que menos llamadas recursivas dan un alto rendimiento.

Tomemos el ejemplo de los números 3 y 7. Si sumamos el número mayor 7 a la variable de suma, entonces,

0+7 = 7 (llamada desde la función principal), 7%3 != 0
7+7 = 14 (primera llamada recursiva), 14%3 != 0
14+7 = 21 (segunda llamada recursiva)
21 es divisible por 3 y 7

Entonces solo habrá 2 llamadas recursivas.De manera similar, si tomamos el número más pequeño (3), entonces,
0+3 = 3 , 3%7 != 0
3+3 = 6 , 6%7 != 0
6+3 = 9 , 9%7 != 0
9+3 = 12 , 12%7 != 0
12+3 = 15 , 15%7 != 0
15+3 = 18 , 18%7 != 0
18+3 = 21
21 es divisible por 3 y 7
Por tanto, habrá 6 llamadas recursivas.

Ejercicio 4
Escribir una función recursiva que devuelva la suma de los primeros N enteros

Ejercicio 5
Programe un método recursivo que transforme un número entero positivo a notación binaria.

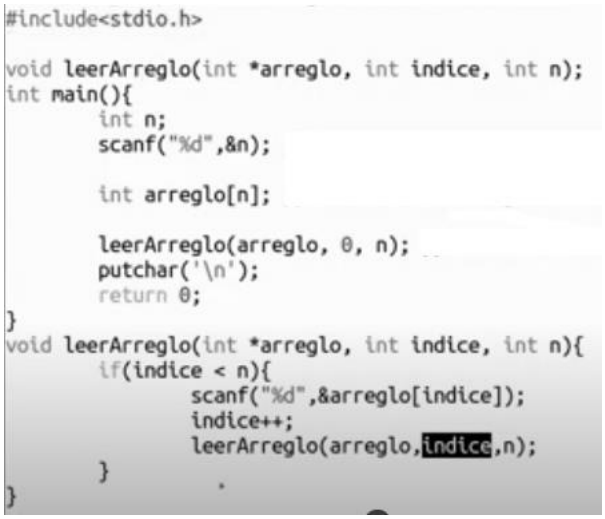
Ejercicio 6
Cuál es el resultado de esta función para distintos valores de X?

```
static int f(int x)
{
                        if (x >100)
                        {
                                return (x-10);
        }
        else
        {
                        return(f(f(x+11)));
```

```
        }
}
```

Ejercicio Nº 7

Para el siguiente programa, elaborar el enunciado.



```c
#include<stdio.h>

void leerArreglo(int *arreglo, int indice, int n);
int main(){
        int n;
        scanf("%d",&n);

        int arreglo[n];

        leerArreglo(arreglo, 0, n);
        putchar('\n');
        return 0;
}
void leerArreglo(int *arreglo, int indice, int n){
        if(indice < n){
                scanf("%d",&arreglo[indice]);
                indice++;
                leerArreglo(arreglo,indice,n);
        }
}
```