❑ **Exercice 1** : *Calcul des termes de la suite de Fibonnacci*
On rappelle que la suite de Fibonacci $(F_n)_{n\in\mathbb{N}}$ est définie par $F_0 = 1$, $F_1 = 1$ et $\forall n \geq 2, F_n = F_{n-1} + F_{n-2}$ (chaque terme est la somme des deux précédents)

1. Donner un algorithme récursif naïf permettant de calculer le nième terme de cette suite.

   Pour calculer F_n, on calcule récursivement F_{n-1} et F_{n-1} et on les ajoute.

2. En proposer une implémentation en OCaml.

```
let rec fibo n =
  if n<2 then 1 else fibo (n-1) + fibo (n-2);;
```

3. On note K_n le nombre d'appels récursifs nécessaires au calcul de F_n. Donner K_0, K_1 et la relation de récurrence vérifiée par $(K_n)_{(n\in\mathbb{N})}$

   $K_0 = 1$, $K_1 = 1$ et $K_n = 1 + K_{n-1} + K_{n-2}$

4. Etablir que $K_n = 2F_n - 1$.

   On effectue une preuve par récurrence de $\mathcal{P}(n) =$ « $K_n = 2F_n - 1$ »
   - $2F_0 - 1 = 1$ et $K_0 = 1$ donc $\mathcal{P}(0)$ est vraie et de même $\mathcal{P}(1)$ est vraie.
   - Soit $n \in \mathbb{N}$ tel que $\mathcal{P}(n)$ et $\mathcal{P}(n-1)$ soient vraies, montrons $\mathcal{P}(n+1)$ vraie.
     $K_{n+1} = 1 + K_n + K_{n-1}$ et par hypothèse de récurrence :
     $K_{n+1} = 1 + 2F_n - 1 + 2F_{n-1} - 1$
     $K_{n+1} = F_n - 1$

5. En déduire la complexité de l'algorithme récursif naïf.

   En notant $\varphi = \dfrac{1+\sqrt{5}}{2}$, on sait que F_n est un grand O de φ^n. Donc l'algorithme récursif naif a une complexité exponentielle ($\varphi > 1$).

6. Donner un algorithme de complexité linéaire permettant de calculer F_n.

   On peut calculer les termes de proches en proches de façon itérative.

7. En fournir une implémentation en langage C.

```
int fibo(int n)
{
    int a = 1;
    int b = 1;
    int t;
    for (int i = 1;i<n;i++)
    {
        t = b;
        b = a + b;
        a = t;
    }
    return b;
}
```

❑ **Exercice 2** : *Suite « lock and say »*

La suite de Conway ou suite « *lock and say* » (regarder et dire) a pour premier terme $u_1 = 1$, puis chaque terme se détermine en énonçant les chiffres du terme précédent. Ainsi,
$u_2 = 11$ (car le terme précédent contient une fois le chiffre 1)
$u_3 = 21$ (car le terme précédent contient 2 fois le chiffre 1)
$u_4 = 1211$ (car le terme précédent contient une fois le chiffre 2 puis 1 fois le chiffre 1)

1. Déterminer les termes u_5, u_6, u_7 et u_8.

   $u_5 = 111221$
   $u_6 = 312211$
   $u_7 = 13112221$
   $u_8 = 1113213211$

2. Proposer une conjecture sur les chiffres pouvant intervenir dans un terme de la suite.

   On conjecture que la suite ne contient que les chiffres 1, 2 ou 3.

3. Prouver cette conjecture

   On effectue une preuve par récurrence sur $\mathbb{N}^*$ de la propriété $\mathcal{P}(n) =$« *"Le nième terme de la suite ne contient que les chiffres 1,2 ou 3* »
   - $\mathcal{P}(1)$ est vraie puisque $u_1 = 1$.
   - Soit $n \in \mathbb{N}$ tel que $\mathcal{P}(n)$ est vraie, montrons qu'alors $P(n+1)$ est vraie. Par hypothèse de récurrence, u_n ne contient que des 1, des 2 ou des 3. La seule façon pour qu'un autre chiffre $k \in \mathbb{N}$, ($k \neq 1, 2$ ou 3) apparaisse dans $P(n+1)$ est donc d'avoir k chiffres consécutifs identiques dans u_n. Or les chiffres de u_{n+1} apparaissent par pairs constitué d'un nombre d'apparitions et d'un chiffre, qu'on note $n_1, c_1, n_2, c_2, \dots n_p, c_p$. Les chiffres de deux pairs consécutives sont différents car sinon ils seraient regroupés dans la même pair, donc $c_i \neq c_{i+1}$ pour $i \in [\![0; p-1]\!]$. Donc cette suite ne peut contenir plus de 3 chiffres consécutifs égaux.

   ⛒ On pourra faire une preuve par récurrence et raisonner par l'absurde.

4. En utilisant le résultat établi à la question précédente, proposer une fonction en OCaml `int list -> int list` qui prend en argument un terme de la suite de Conway (sous la forme de la liste de ses chiffres) et renvoie le terme suivant (toujours sous la forme de la liste de ses chiffres)

   D'après la résultat précédent on ne peut avoir plus de 3 chiffres consécutifs égaux, on peut donc se limiter à une correspondance de motifs sur au maximum 3 chiffres :

   ```
   let rec lockandsay previous =
     match previous with
     | [] -> []
     | x::[] -> [1;x]
     | x::y::[] -> if x=y then [2;x] else [1;x;1;y]
     | x::y::z::t -> if (x=y && y=z) then 3::x::lockandsay(t) else
                     if x=y then 2::x::lockandsay(z::t) else
                       1::x::lockandsay(y::z::t)
   ```

   ⛒ Utiliser une correspondance de motif sur les chiffres par groupe de 3.

5. Ecrire une fonction en OCaml utilisant la fonction précédente et calculant le nième terme de la suite de Conway.

```
let rec ls n =
  let rec aux n current =
    if n=1 then current else aux (n-1) (lockandsay current) in
    aux n [1];;
```