TD N°4
Les fonctions

1. Suite de Fibonacci

La suite de Fibonacci est une suite de nombres dont chaque terme est la somme des deux précédents. Les deux premiers termes de la suite sont par définition égaux à 1.

$$\begin{cases} F_1 = 1 \\ F_2 = 1 \\ F_n = F_{n-1} + F_{n-2} \quad \text{pour} \quad n > 2 \end{cases}$$

Ecrire une fonction permettant de calculer le nième terme de cette suite.

Corrections :

On montre que le calcul direct est plus rapide que par récurrence

```
int fibonacci(int n){
	int i,u,v,w;
	u = 1;
	v = 1;
	for(i=3;i<=n;i++){
		w = u+v;
		u = v;
		v=w;
	}

	return v;
}
```

Etat	i	u	v	w
init	x	1	1	x
#1	1	1	2	2
#2	2	2	3	3
#3	3	3	5	5
#4	4	5	8	8

2. Puissance d'un nombre

A) Ecrivez une fonction *puissance* qui calcule x^n pour *x* flottant quelconque et *n* entier positif ou nul. Calculez le nombre d'opérations effectuées, déduisez-en la complexité de la fonction.

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

float puissance(float x, int n){
      int i = 0;
      float res = 1.0;
      if( n < 0){
            perror("n doit etre postif ou nul\n");
            exit(EXIT_FAILURE) ;
      } else if( n >= 38 ){
            perror("n est trop grand !\n");
            exit(EXIT_FAILURE) ;

      } else{

            for(i=0;i < n;i++){
                  res *= x;
            }
            return(res);
      }
}
```

n multiplications et n affectations => complexité linéaire : O(n)

B) Ecrivez une nouvelle fonction puissance en mettant à profit la remarque suivante :

$$x^n = \begin{cases} x \times x^{n-1} & \text{si n est impair} \\ (x^2)^{\frac{n}{2}} & \text{si n est pair} \end{cases}$$

Calculez le nombre d'opérations effectuées, en déduire la complexité de cette nouvelle fonction.

Solution :

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

float puissance(float x, int n){
	int i = 0;
	float res = 1.0;
	if( n < 0){
		perror("n doit etre postif ou nul\n");
		exit(EXIT_FAILURE) ;
	} else if( n >= 38 ){
		perror("n est trop grand !\n");
		exit(EXIT_FAILURE) ;

	} else if(  n == 0){
		return(1.0);
	} else if(  n == 1){
		return(x);
	} else if(  n%2 != 0){ // n est impair
		return(n*puissance(x,n-1));

	} else { // n est pair
		return(puissance(x*x,n/2));

	}
}
```

Log2(n) multiplications et log2(n) divisions => complexité logarithmique : O(log(n))

3. Produit Matriciel

Ecrivez une fonction *produitMat* qui effectue le produit d'une matrice réelle *A* ayant *nla* lignes et *nca* colonnes par une matrice réelle *B* ayant *nlb* lignes et *ncb* colonnes. Le résultat sera stocké dans une matrice *C*.

On rappelle la définition suivante : soient $A \in \mathbb{R}^{nla \times nca}$ et $B \in \mathbb{R}^{nlb \times ncb}$ on a $C = A \cdot B$ avec $C \in \mathbb{R}^{nla \times ncb}$ tel que :

$$C_{ij} = \sum_{k=1}^{nca} A_{ik} \cdot B_{kj}$$

Solution :

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

void produitMat(float* A, int nla, int nca, float* B, int nlb, int ncb,
float* C){
	int i,j,k;
	int nlc,ncc;
	float acc;
	if( nca != nlb){
		perror("nla doit etre egal a nlb\n");
		exit(EXIT_FAILURE);
	}else if( (nla <= 0) || (nca <= 0) || (nlb <= 0) || (ncb <= 0)){
		perror("nla, nca, nlb et ncb doivent etre positfs ou nuls\n");
		exit(EXIT_FAILURE);
	}else{
		nlc = nla;
		ncc = ncb;
		for(i=0; i<nlc; i++){
			for(j=0; j<ncc; j++){
				acc = 0.0;
				for(k=0;k<nca;k++){
					acc += A[k+i*nca]*B[j+k*ncb];
				}
				C[j+i*ncc] = acc;
			}
		}
	}

}
```