TP 2 – Fonctions en C

Exercice 1 – triangle de Pascal

1. Ecrire une fonction itérative de prototype ***long fact (int n)*** qui retourne la factorielle d'un entier donné en argument.
2. Ecrire une fonction de prototype ***long binome(int n, int k)*** pour calculer $C_n^k = \frac{n!}{k! \times (n-k)!}$
3. Ecrire une fonction de prototype ***void printPascal (int n)*** pour afficher un triangle de Pascal de n lignes.
4. Ecrire la fonction principale ***main*** pour lire n puis afficher un triangle de Pascal de n lignes. Vous pouvez répéter la lecture et l'affichage dans une boucle.

Exercice 2 – un peu d'aléatoire

Le langage C dispose d'un générateur de nombres pseudo-aléatoire, c'est la fonction rand déclarée dans stdlib.h. Ce générateur est initialisé à l'aide de la fonction srand par le temps en secondes écoulé depuis le 1 janvier 1970 (obtenu avec la fonction time déclarée dans time.h). Le nombre généré par rand après chaque appel est compris entre 0 et une constante prédéfinie RAND_MAX.

Exemple :

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

main(){
    int a, i;

    //initialisation du générateur
    srand(time(NULL));

    //génération de 4 entiers aléatoires entre 0 et RAND_MAX
    for (i=1; i<=4; i++){
        a = rand();
        printf("a = %d\n", a);
    }
}
```

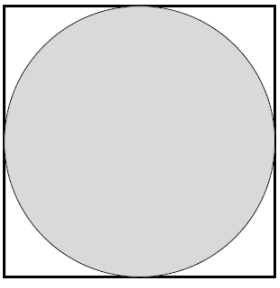
Pour générer un nombre réel aléatoire entre 0 et 1 :

```
double X =((double)rand()/(double)RAND_MAX);
```

1. Ecrire une fonction ***entierAleatoire (int min, int max)*** qui utilise la fonction rand pour générer et retourner un entier aléatoire dans l'intervalle [min, max].
2. Ecrire une fonction similaire ***reelAleatoire(double min, double max)***, qui cette fois ci génère un réel dans l'intervalle [min, max].
3. Ecrire une fonction ***simulationDe (int n)*** qui simule la lancée d'un dé n fois, et qui affiche la fréquence de chaque valeur de 1 à 6.

Méthode de Monté Carlo :

On peut calculer la surface d'un cercle on utilisant une simulation de Monté Carlo. Pour cela on génère ***n*** points de coordonnées (x, y) aléatoires dans le rectangle $[-r, r]^2$, et on compte le nombre de points qui sont dans le cercle de rayon r.



Dans ce cas, on a l'approximation suivante :

$$\frac{Nombre\ de\ points\ dans\ le\ cercle}{n} \approx \frac{Surface\ du\ cercle}{Surface\ du\ carré}$$

4. Ecrire une fonction ***pointSuivant (double r)*** qui génère deux réels x et y dans l'intervalle [-r, r], puis renvoie 1 si le point (x, y) est dans le cercle, et 0 sinon.
5. Ecrire une fonction ***surfaceCercle (double r, int n)*** qui retourne la surface d'un cercle en utilisant la méthode de Monté Carlo, n étant le nombre de point générés. Exécuter la fonction et remarquer que plus le nombre de points est grand plus l'approximation de la surface est meilleure.
6. Utiliser cette fonction pour calculer une approximation du nombre π.

Exercice 3 – récursivité

1. Ecrire une fonction *récursive* ***long fibo(int n)*** qui retourne la valeur de n^{ième} terme de la suite de Fibonacci.

On se propose de compter le nombre d'appels récursifs de la fonction pour un n donné. Pour cela :

Ajouter une variable globale NR pour calculer le nombre d'appels de cette fonction.

Initialiser cette variable par 0 (d'ailleurs c'est sa valeur par défaut)

Puis modifier la fonction fibo pour qu'elle incrémente NR après chaque appel.

2. Ecrire la fonction principale main pour calculer les n premiers termes de cette suite de Fibonacci, en affichant pour chaque terme i, fibo (i) et le nombre d'appels nécessaires pour calculer fibo (i).

Exemple : pour n = 5, on affiche :

i	fibo(n)	NR
0	1	1
1	1	1
2	2	3
3	3	5
4	5	9
5	8	15

Exécutez la fonction pour n = 10, 20, 30, … et remarque l'évolution du nombre d'appels récursifs.
Pour réduire le nombre d'appels récursifs, on propose une nouvelle implémentation de la suite qui se base sur l'idée suivante :

Au lieu d'utiliser la formule $\begin{cases} U_0 = 1 \\ U_1 = 1 \\ U_{n+1} = U_n + U_{n-1} \end{cases}$ on utilise la formule $\begin{cases} V_0 = U_1 \\ V_1 = U_1 + U_0 \\ V_{n+1} = V_n + V_{n-1} \end{cases}$

3. Ecrire alors une fonction ***long sommeAdditive (int n , int u0, int u1)*** qui utilise la récurrence pour évaluer le n^{ième} terme de la suite de Fibonnaci.
4. Reprendre la question 2 avec cette nouvelle implémentation de la suite, et remarquer le nombre d'appels récursifs.

Exemple : pour n = 5, on doit avoir :

i	fibo(n)	NR
0	1	1
1	1	2
2	2	3
3	3	4
4	5	5
5	8	6

Exercice 4 – récursivité et jeu de Tours de Hanoï

Le jeu des Tours de Hanoï est comme suit : On a 3 piquets en face de soi, numérotés 1, 2 et 3 de la gauche vers la droite, et n rondelles de tailles toutes différentes entourant le piquet 1, formant un cône avec la plus grosse en bas et la plus petite en haut. On veut amener toutes les rondelles du piquet 1 au piquet 3 en ne prenant qu'une seule rondelle à la fois, et en s'arrangeant pour qu'à tout moment il n'y ait jamais une rondelle sous une plus grosse.
Un raisonnement par récurrence permet de trouver la solution en quelques lignes.
Ecrire un programme en C pour résoudre ce problème.