Informatique *TD*
Programmation impérative avec C - Pointeurs, tableaux, structures Ipesup

L'objectif est de programmer en **C** un jeu de Pierre–Feuille–Ciseaux entre l'ordinateur et l'utilisateur. On jouera d'abord une manche, puis plusieurs. On maintiendra un score final pour déterminer le vainqueur.

Exercice 1 : Préliminaires — générateur pseudo-aléatoire

Dans cet exercice, on (re)voit la génération d'entiers aléatoires en C avec `rand()`.

1) Créez un fichier `de.c` et écrivez le code suivant :

```c
// de.c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

void affiche_lance(void) {
    int n = (rand() % 6) + 1; // entier entre 1 et 6
    printf("%d\n", n);
}

int main(void) {
    srand(time(NULL)); // initialisation du generateur
    affiche_lance();
    return 0;
}
```

2) Compilez et exécutez plusieurs fois le programme. Que fait `affiche_lance` ? Quelles valeurs peuvent être affichées ?
3) Modifiez `affiche_lance` pour qu'elle prenne un **paramètre optionnel conceptuel** k (en C, écrivez une `affiche_lance_k(int k)`) et qu'elle affiche le résultat d'un dé équitable à k faces (numérotées de 1 à k). Par défaut, $k = 6$.
4) Dans le `main`, écrivez une boucle qui lance un dé à 9 faces 42 fois et affiche les 42 résultats.
5) Écrivez une fonction `affiche_somme_lances(int n)` qui simule n lancers d'un dé à 6 faces et affiche la somme. Quelle valeur obtenez-vous pour `affiche_somme_lances(1000)` ? Cela suggère-t-il une distribution particulière ?

Exercice 2 : Jeu en une manche

On crée un jeu de Pierre–Feuille–Ciseaux en une manche. On n'utilise **pas** de tableaux ni de chaînes ; on représente les coups avec un caractère unique :

$$\texttt{'P'} = \text{Pierre}, \quad \texttt{'F'} = \text{Feuille}, \quad \texttt{'C'} = \text{Ciseaux}.$$

1) Écrire une fonction `convert(int n)` qui renvoie un `char` selon n : si $n = 1$ retourner `'P'`, si $n = 2$ retourner `'F'`, sinon retourner `'C'`.

2) Écrire une fonction `tirage(void)` qui retourne le choix aléatoire de l'ordinateur.

> `tirage()` retourne aléatoirement l'un des caractères `'P'`, `'F'` ou `'C'`.

3) Tester `tirage()` un grand nombre de fois (boucle) pour vérifier qu'elle semble uniforme.
4) Écrire une fonction `coupJoueur(void)` qui utilise `scanf(" %c", &c)` pour demander le choix de l'utilisateur et qui **valide** l'entrée (re-demander tant que ce n'est pas `'P'`, `'F'` ou `'C'`).

> `coupJoueur()` demande `P/F/C` à l'utilisateur et retourne le caractère valide.

5) Écrire une fonction `uneManche(void)` qui fait jouer une manche entre l'utilisateur et l'ordinateur. Le programme affiche les coups et le résultat (victoire joueur, victoire ordinateur, égalité) en se basant sur les règles : `P` bat `C`, `C` bat `F`, `F` bat `P`.

Exercice 3 : Jeu en plusieurs manches

1) En se basant sur `uneManche`, écrire une fonction `manche(void)` qui fait jouer une manche et **retourne deux caractères** :
   — le premier est le coup du joueur parmi `'P','F','C'` ;
   — le second est le résultat : `'J'` (joueur gagne), `'O'` (ordinateur gagne) ou `'E'` (égalité).

> `manche()` renvoie par exemple `'P','J'` si le joueur a joué Pierre et a gagné, `'F','O'` si l'ordinateur gagne, `'C','E'` s'il y a égalité.

2) Écrire une procédure `chifoumi(int n)` qui joue n manches, affiche le vainqueur final, son nombre de victoires et un **résumé** des manches sous forme d'une *ligne* : par exemple `P:J P:O P:E P:J C:J` (on affiche `Coup:Résultat` pour chaque manche).

Exercice 4 : Pour aller plus loin — Variante Lézard–Spock

Modifier le programme pour ajouter l'option Pierre–Feuille–Ciseaux–Lézard–Spock. On **garde** les règles précédentes et on ajoute : le lézard mange le papier, empoisonne Spock, est écrasé par la pierre, est décapité par les ciseaux. Spock vaporise la pierre, casse les ciseaux, et est discrédité par le papier.

Représentation : toujours un caractère unique, en ajoutant `'L'` (Lézard) et `'S'` (Spock).

Essayez de **réutiliser** au maximum les fonctions déjà écrites (pas de recodage inutile).