Feuille 4 : fonctions

Fonctions et prototypes : Dans l'ensemble de cette feuille d'exercices, vous devrez réaliser des fonctions. Si vous ne parvenez pas à écrire une fonction, commencez par son prototype.
Générateur aléatoire (rappel) : En langage C, la fonction `rand` de la bibliothèque `stdlib.h` retourne "aléatoirement" une valeur de type `int` entre 0 et un grand entier `RAND_MAX`. Ainsi, pour que l'ordinateur génère aléatoirement un entier $\texttt{n} \in [5, 20]$ par exemple, on peut procéder de la façon suivante :

- inclure les bibliothèques `stdlib.h` et `time.h`,
- appeler `srand(time(NULL)) ;` en début de programme (afin d'initialiser le générateur),
- récupérer le nombre aléatoire `n` en faisant `n = rand()%16+5 ;`. En effet : on veut un entier quelconque entre 5 et 20 (compris). Or, entre 5 et 20, il y a 16 entiers c'est pourquoi on considère `rand()%16`. Toutefois, le résultat de cette formule est un entier entre 0 et 15. Pour obtenir un entier entre 5 et 20, il reste à ajouter 5.

Exercice 1 : *(TD)*

1. Écrire une fonction `syracuse` qui, pour un entier n donné en paramètre et supposé strictement positif, retourne la valeur de la fonction de Syracuse en n. On rappelle que la fonction de Syracuse est définie sur $\mathbb{N}^*$ par :
$$f(1) = 1, \quad f(n) = \begin{cases} n/2 & \text{si } n \text{ est pair} \\ 3n+1 & \text{si } n \text{ est impair et différent de 1.} \end{cases}$$
2. Écrire une fonction `saisieN` qui retourne une valeur entière strictement positive saisie au clavier. La saisie sera effectuée avec contrôle (de validité) ce qui signifie qu'une valeur est demandée tant que l'utilisateur entre une valeur inférieure ou égale à 0.
3. Écrire une fonction `trouveUnEnM` qui, pour un entier n donné en paramètre et supposé strictement positif, retourne le plus petit entier m tel que $f^m(n) = (f \circ f \cdots \circ f)(n) = 1$.
4. Écrire une fonction `afficheM` qui, pour chaque entier $n \in$ [`nMin`,`nMax`] (les paramètres de la fonction, `nMin` et `nMax`, étant supposés strictement positifs), affiche la valeur de n suivi du petit entier m tel que $f^m(n) = 1$.
5. Écrire un programme appelant les fonctions écrites questions 2 et 4.

Exercice 2 : *(TD+TP)*

On considère la suite de Fibonacci $(F_n)_{n\in\mathbb{N}}$ définie par les relations suivantes :
$$\begin{cases} F_0 = 0, \quad F_1 = 1, \\ F_{n+2} = F_n + F_{n+1} \quad \text{pour } n = 0,\, 1,\, 2, \ldots \end{cases}$$
On rappelle que la suite de Fibonacci est strictement croissante et qu'elle tend vers $+\infty$.

1. On considère les instructions en langage C suivantes, concernant des variables entières :
```
tmp = f0 ; f0 = f1 ; f1 = tmp+f0 ;
```
Si avant l'exécution de cette ligne, `f0` contient l'entier x et `f1` l'entier y, quels seront les contenus de `f0` et `f1` après l'exécution des trois instructions ?
2. Ecrire une fonction qui, à partir de deux entiers `a` et `b` (avec $2 \leq$`a`$\leq$`b`), retourne le nombre de termes de la suite de Fibonacci appartenant à l'intervalle [`a`,`b`].
3. Ecrire une fonction qui, à partir d'un entier `m` (avec $2 \leq$`m`), affiche les `m` premiers termes impairs (pas d'indice impair) de la suite de Fibonacci.

4. Ecrire un programme permettant :
   a) la saisie de deux entiers a, b (on supposera que a et b vérifient $2 \leq a \leq b$) et l'affichage du nombre de termes de la suite de Fibonacci appartenant à l'intervalle $[a, b]$,
   b) la saisie d'un entier m vérifiant $2 \leq m$ et l'affichage des m premiers termes impairs de la suite de Fibonacci.

   Vous pourrez faire une fonction `saisie` qui lit un nombre entier au clavier supérieur à un entier *min*. Tant que le nombre entré par l'utilisateur ne satisfait pas cette propriété, un nouveau nombre doit être entré. Cette fonction sera utilisée pour entrer a, b et m.

Exercice 3 : *(TD+TP)*

Un cavalier est au départ d'un parcours de saut d'obstacles, composé de `nObstacles` obstacles à franchir.

a) Pour chaque obstacle, le cheval a 7 chances sur 10 de franchir l'obstacle (le résultat du franchissement sera déterminé aléatoirement),

b) Pour le premier obstacle, il ne franchira pas l'obstacle dans 3 cas sur 10 et devra recommencer,

c) Pour chacun des autres obstacles, il fera tomber une barre dans 2 cas sur 10 et devra tenter une nouvelle fois de franchir cet obstacle, ou refusera de sauter dans 1 cas sur 10 et devra repartir à l'obstacle précédent.

1. Écrire une fonction retournant un entier aléatoire entre 1 et 10.
2. Écrire une fonction affichant la position `pos` du cheval sur un parcours à `nObstacles` obstacles sous la forme suivante dans le cas où `pos`=5 (5 obstacles ont été franchis) et `nObstacles`=12 :

   ```
   -|-|-|-|-|o|-|-|-|-|-|-|-
   ```

3. Écrire une fonction simulant l'évolution complète d'un parcours à `nObstacles` obstacles (une ligne sera affichée après chaque tentative de saut d'un obstacle) et retournant le nombre de tentatives de sauts effectuées pour faire le parcours,
4. Écrire un programme permettant de demander le nombre d'obstacles du parcours, de suivre l'évolution complète sur ce parcours et d'afficher le nombre de tentatives de sauts.
5. Compléter le programme précédent afin de simuler plusieurs passages sur le même parcours (le nombre de passages étant demandé à l'utilisateur), et affichant le nombre moyen de tentatives de sauts.

Exercice 4 : *(TD)*

Le but est de programmer le calcul de x^n où x est un réel non nul et n un entier positif. Pour cela, on utilise le fait que :

$$x^0 = 1, \quad x^p = x.x^{p-1} \text{ si } p \in \mathbb{N}^*. \tag{1}$$

On demande d'écrire un programme complet qui saisit un réel non nul `x` et un entier positif `n`, puis qui calcule $\texttt{x}^{\texttt{n}}$ à l'aide de chacune des fonctions proposées ci-dessous :

1. Écrire une première fonction non récursive qui retourne la valeur $\texttt{x}^{\texttt{n}}$ calculée à partir de (1).
2. Écrire une deuxième fonction récursive qui retourne la valeur $\texttt{x}^{\texttt{n}}$ calculée à partir de (1).

Remarque : Il existe une fonction `pow` de la bibliothèque `math.h` : `pow(x,y)` vaut x^y quand cette quantité est définie.