Récursivité

1 Introduction à la récursivité

Exercice 1. Introduction à la récursivité.

On cherche à réaliser un parallèle entre la notation mathématique et la rédaction en C d'une fonction récursive. Prenons l'exemple d'une fonction calculant la somme des entiers de 1 à n à partir d'un entier positif n donné. Soit f la notation mathématique d'une telle fonction et `SommeRec` sa notation en C.

1) Explicitez la notation mathématique de f (ensembles de départ, d'arrivée ...).
2) Rédigez le prototype de la fonction `SommeRec`.
3) Montrez que $f(n) = f(n-1) + n$ pour tout entier n positif.
4) Rédigez l'appel récursif de la fonction `SommeRec`.
5) Complétez la fonction `SommeRec` avec la condition d'arrêt.

Exercice 2. Récursivité simple : puissance.

Considérons le prototype suivant.

```
unsigned long puissanceRec (unsigned long a, short n);
```

Implémentez cette fonction calculant l'entier `a` élevé à la puissance `n` de façon récursive.

Exercice 3. Récursivité simple : division euclidienne.

Rappelons que la division euclidienne d'un entier a par un entier b cherche à obtenir les entiers q et r tels que $a = bq+r$ avec $0 \leq r < b$. Ce résultat peut être obtenu par soustractions successives de b à l'entier a. Implémentez donc la fonction récursive `DivEucRec` prenant en entrée des entiers a et b et donnant le reste de la division euclidienne de a par b par soustractions successives.

2 Récursivité multiple

Exercice 4. Récursivité multiple : triangle de Pascal.

L'objectif de l'exercice est d'afficher les n premières lignes du triangle de Pascal. Pour cela, on rappelle que la case C de la ligne i et de colonne j s'obtient par :

$$C_{i,j} = C_{i-1,j} + C_{i-1,j-1}\,.$$

Voici par exemple les 6 premières lignes du triangle de Pascal :

```
1
1   1
1   2   1
1   3   3   1
1   4   6   4   1
1   5   10  10  5   1
```

Écrivez les fonctions suivantes.

1) `int caseTriangle (int NumLig, int NumCol)` qui retourne la valeur, calculée de manière récursive classique en utilisant la formule vue ci-dessus, de la case d'indice de ligne `NumLig` et de colonne `NumCol` du triangle de Pascal. On considère que les indices `NumLig` et `NumCol` commencent à 0.
2) `void afficheTriangle (int NbLignes)` qui affiche les `NbLignes` premières lignes du triangle de Pascal. Chaque case du triangle doit être séparée par un espace lors de l'affichage. Cette fonction itérative doit appeler la fonction `caseTriangle()`.

3 Récursivité mutuelle

Exercice 5. Je suis ton pair !.

L'objectif de cet exercice est de déterminer la parité d'un entier positif à l'aide de la récursivité mutuelle. La complexité est extrêmement mauvaise, l'implémentation n'est pas du tout naïve et il existe déjà l'opérateur modulo (`%`) qui fait parfaitement son travail. Mais c'est rigolo donc ...

En sachant que 0 est un nombre pair, et 1 est un nombre impair, codez deux fonctions `estPair` et `estImpair` prenant en entrée un entier positif et renvoyant 1 si l'entier donné en argument est pair (respectivement impair), 0 sinon. Ces fonctions doivent être récursives mutuelles. N'oubliez pas que si un entier n est pair, $n-1$ est impair.

Exercice 6. Modèle proies-prédateurs.

Les équations de Lotka-Volterra introduites en 1925 et 1926 permettent de simuler simplement l'évolution de population d'individus selon un modèle proie prédateur. Dans ce modèle, les proies se développent naturellement grâce à une ressource illimitée. Ils ne meurent qu'à cause de leurs prédateurs qui eux ne se développent que grâce à la profusion des proies. Ce modèle peut être mis en place pour illustrer l'évènement survenu durant la Première Guerre mondiale en mer Adriatique.

Avant le début de la guerre, la pêche intensive des sardines permettait de trouver un équilibre entre les populations de sardines et de requins. Les sardines se développaient grâce au plancton en ressource illimitée, et les requins mangeaient alors les sardines. Durant la guerre, il n'y a pas eu de pêche et la population des sardines a alors explosé. Au retour de la guerre, les pêcheurs reviennent, et ont alors la mauvaise surprise de ne pêcher que très peu de sardines ! Comment expliquer ce phénomène ?

Sans prédateur, la population de sardines évolue selon l'équation $S_n = (1+a)S_{n-1}$, avec a un réel strictement positif permettant de décrire l'augmentation (exponentielle) de la population de sardines. Sans nourriture les requins disparaissent suivant la formule $R_n = (1-k)R_{n-1}$, avec k un réel strictement positif. Lorsque les deux espèces cohabitent, les sardines disparaissent à cause des requins, et perdent à chaque instant t_n $bS_{n-1}R_{n-1}$ individus, un nombre proportionnel à la population de requins et de sardines à l'instant t_{n-1}. De façon analogue, la population de requins augmente de $cS_{n-1}R_{n-1}$ individus à chaque instant.

Les équations de Lotka-Volterra illustrent alors ces populations pour un instant t_n donné :

$$\begin{cases} S_{n+1} &= S_n(1+a-bR_n) \\ R_{n+1} &= R_n(1-k+cS_n) \end{cases} \tag{1}$$

Vous devez implémenter deux fonctions récursives permettant de calculer le nombre de requins et le nombre de sardines à un instant donné. Ces fonctions sont récursives mutuelles et utilisent les formules de Lotka-Volterra vues ci-dessus. Vous pouvez vérifier votre code avec Libre Office, en réalisant une feuille de calcul qui vous permettra de tracer les graphes de ces deux populations au cours du temps. Vous pouvez utiliser les valeurs initiales suivantes pour tester votre code : $a = 0,03$ qui est le taux de croissance de la population de sardines, $b = c = 0,0005$ qui sont les facteurs d'évolution d'une population en fonction de l'autre population et $k = 0,06$ qui est le taux de mortalité des requins.

4 Pour aller plus loin

Exercice 7. Euclide forever.

Comme vous le savez, l'algorithme d'Euclide permet calculer rapidement le PGCD de deux entiers. Pour rappel, voici les différentes étapes de cet algorithme appliqué aux entiers 119 et 544.

$$\begin{aligned} 119 &= 544 \times 0 + 119 \\ 544 &= 119 \times 4 + 68 \\ 119 &= 68 \times 1 + 51 \\ 68 &= 51 \times 1 + \mathbf{17} \\ 51 &= 17 \times 3 + 0 \end{aligned}$$

Le PGCD est le dernier reste non nul, ici 17.

1) Cet algorithme vous est donné de manière itérative ci-dessous. À vous de le modifier pour en faire un algorithme récursif.

Algorithme 1 – Euclide
Entrée. Entiers naturels a, b
Sortie. Entier naturel PGCD(a,b)

```
Entiers r, pgcd
pgcd ← b
Faire
Faire
    r ← reste de la division euclidienne de a par b
    Si r ≠ 0 alors
        pgcd ← r
    Fin Si
    a ← b
    b ← r
Tant que r ≠ 0
Renvoyer pgcd
```

2) Implémentez votre algorithme en langage C.

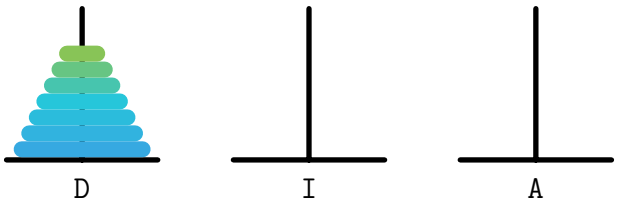
Exercice 8. It's Hanoï-ing.

Le but de cet exercice est de résoudre le problème des tours de Hanoï. Le jeu consiste à déplacer `N` disques de diamètres différents d'une tour de départ `D` vers une tour d'arrivée `A` en utilisant une tour intermédiaire `I`. Pour arriver à cela, il faut respecter les deux règles suivantes :

- on ne peut déplacer qu'un disque à la fois, du sommet d'une tour vers le sommet d'une autre tour ;

- on ne peut déplacer un disque que sur un disque de diamètre supérieur ou sur une tour vide.

Initialement, les `N` disques sont sur la tour de départ, de plus grand au plus petit.



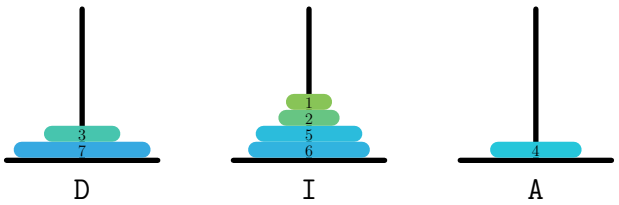
Pour résoudre ce problème, on s'appuiera sur l'algorithme `Hanoi(n, T1, T2, T3)`.

- Lors de l'exécution de `Hanoi(n, T1, T2, T3)` (avec $0 \leq n \leq N$), on suppose que les `n` plus petits disques sont empilés au sommet de la tour `T1`, les `N - n` plus grands disques peuvent être répartis sur les autres tours.
- À l'issue de la fonction, la pile des `n` plus petits disques se retrouve au sommet de la tour `T3`, les `N - n` plus grands disques n'ont pas été déplacés.

Ainsi, l'exécution de `Hanoi(N, D, I, A)` permet de résoudre le problème.

1) Expliciter l'algorithme `Hanoi(n, T1, T2, T3)`.
2) Appliquer votre algorithme pour résoudre le jeu de Hanoï avec `N = 3` tours.

On modélise maintenant une tour comme un tableau de `N` cases. Le disque de plus petit diamètre est représenté par l'entier 1, le plus grand par l'entier `N`. Les emplacements vides dans une tour sont représentés par la valeur −1. Par exemple,



est représenté par

```
D = [3,  7 , -1, -1, -1, -1, -1]
I = [6,  5 ,  2 ,  1 , -1, -1, -1]
A = [4, -1, -1, -1, -1, -1, -1]
```

3) Modifier votre algorithme pour qu'il fonctionne avec cette modélisation.
4) L'implémenter en C pour qu'il affiche la résolution du problème des tours de Hanoï.

Exercice 9. Ils sont fous ces Romains !.

Le but de cet exercice est de convertir un nombre écrit en chiffres romains en un nombre en chiffres arabes. Pour cela, convenons des valeurs suivantes :

Chiffre romain	M	D	C	L	X	V	I
Valeur	1000	500	100	50	10	5	1

Pour convertir un nombre romain en nombre arabe, il faut le lire « lettre par lettre ». Deux cas sont possibles :

- le chiffre romain courant est inférieur au suivant : on le soustrait de la valeur totale (par exemple XIV qui vaut 14) ;
- le chiffre romain courant est supérieur au suivant : on l'ajoute à la valeur totale (par exemple X1 qui vaut 11).

1) Écrire un algorithme afin de résoudre ce problème de manière récursive.
2) Implémenter votre algorithme en C.