# TP 7 : Programmation de fonctions récursives

On utilisera la ligne de commande suivante pour compiler :

```shell
gcc -Wall -Wextra -Werror -Wvla -fsanitize=address,undefined -o tp7.exe tp7.c
```

S'il est naturel d'écrire des fonctions récursives pour manipuler des structures de données récursives (listes, arbres, etc.), il est aussi possible d'utiliser la récursion pour résoudre d'autres problèmes, dont certains qui ne semblent pas avoir de rapport avec la récursion au premier abord.

## Exercice 1 – Suites et quantités définies par récurrence

On pose la suite $(u_n)$ définie par $u_0 = 1$ et pour tout $n \geq 0$, $u_{n+1} = 3 \times u_n + 2$.

▶ **Question 1** Écrire une fonction récursive `int u(int n)` qui calcule $u_n$ en utilisant la définition de la suite par récurrence et en l'écrivant récursivement.

Souvent, on peut transformer une fonction récursive en une fonction itérative, et vice-versa.

▶ **Question 2** Écrire une fonction itérative `int u_iter(int n)` qui calcule $u_n$ sans utiliser la récursion.

▶ **Question 3** Reprendre le principe de la fonction `int PGCD(int a, int b)` vue dans le TP 4 et écrire une fonction récursive `int PGCD_rec(int a, int b)` qui calcule le PGCD de deux entiers positifs $a$ et $b$ en utilisant la méthode d'Euclide.

On définit la suite de Fibonacci comme suis :

$$\begin{cases} F_0 = 0 \\ F_1 = 1 \\ F_n = F_{n-1} + F_{n-2} \quad \text{pour } n \geq 2 \end{cases}$$

▶ **Question 4** Écrire une fonction récursive `int fib_rec(int n)` qui calcule $F_n$ en utilisant la définition de la suite par récurrence et en l'écrivant récursivement.

▶ **Question 5** Tester cette fonction en augmentant progressivement la valeur de $n$ jusqu'à $n = 45$. Que constate-t-on ? *Pro tip : pour arrêter un programme lancé dans le shell sous Linux, on peut utiliser la combinaison de touches `Ctrl+C`.*

Pour régler le problème de cette première version, on peut voir la suite de Fibonnacci comme deux suites mutuellement récurrentes :

$$\begin{cases} a_0 = 0 \\ a_{n+1} = b_n \end{cases} \quad \text{et} \quad \begin{cases} b_0 = 1 \\ b_{n+1} = a_n + b_n \end{cases}$$

★ **Question 6** Montrer par récurrence que pour tout $n \geq 0$, $a_n = F_n$ et $b_n = F_{n+1}$.

Pour représenter un couple, on peut utiliser le type suivant :

```c
typedef struct {int fst; int snd;} Couple;
```

▶ **Question 7** Écrire une fonction récursive `Couple fib_couple(int n)` qui calcule le couple $c_n$ en utilisant la définition de la suite par récurrence et en l'écrivant récursivement. Tester cette fonction en augmentant progressivement la valeur de $n$ jusqu'à $n = 100$. Que constate-t-on ?

★ **Question 8** Adapter cette fonction pour qu'elle soit itérative. On pourra, ou non, utiliser le type `couple`.

## Exercice 2 – Chemins dans une grille

Supposons que l'on dispose d'une grille infinie, dont chaque case est identifiée par un couple d'entiers naturels. Dans chaque case, on peut se déplacer soit vers la gauche (en soustrayant 1 à la première coordonnée), soit vers le bas (en soustrayant 1 à la seconde coordonnée). On ne peut pas sortir de la grille, donc on ne peut pas se déplacer à gauche si on est déjà dans la colonne 0, et on ne peut pas se déplacer vers le bas si on est déjà dans la ligne 0.
L'objectif est de se déplacer depuis la case $(m, n)$ jusqu'à la case $(0, 0)$, et de compter le nombre de façons différentes d'y parvenir.

▶ **Question 1** Pour quelles valeurs de $(m, n)$ n'y a-t-il qu'une seule façon de se déplacer jusqu'à la case $(0, 0)$ ? *Cela constituera le cas de base de notre fonction récursive.*

▶ **Question 2** Exprimer $C(m, n)$ en fonction de $C(m-1, n)$ et $C(m, n-1)$ dans les autres cas.

▶ **Question 3** Écrire une fonction récursive `int nb_chemins(int m, int n)` qui calcule le nombre de façons différentes de se déplacer depuis la case $(m, n)$ jusqu'à la case $(0, 0)$.

## Exercice 3 – Calculs de coefficients binomiaux

Vous connaissez la notation $\binom{n}{k}$, qui se lit « $n$ parmi $k$ » et qui désigne le nombre de façons de choisir $k$ éléments parmi $n$. Par exemple, $\binom{5}{2} = 10$ car il y a 10 façons de choisir 2 éléments parmi 5. Vous connaissez aussi probablement la formule du triangle de Pascal :

$$\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k}$$

avec les cas de base $\binom{n}{0} = 1$ et $\binom{n}{n} = 1$.

▶ **Question 1** Écrire une fonction récursive `int binom(int n, int k)` qui calcule $\binom{n}{k}$ en utilisant la formule du triangle de Pascal. Une exception doit être levée si on n'a pas $0 \leq k \leq n$.

▶ **Question 2** Tester cette fonction en augmentant progressivement la valeur de $n$ jusqu'à $n = 45$.

▶ **Question 3** Dessiner l'arbre d'appels récursifs de l'appel `binom(5, 2)`. Que constate-t-on ?

★ **Question 4** Exprimer le nombre d'appels récursifs effectués par `binom(n, k)` en fonction de $n$ et $k$, et implémenter une fonction `int nb_appels_binom(int n, int k)` qui calcule ce nombre.

★ Une première méthode pour améliorer l'efficacité de cette fonction est de stocker les résultats déjà calculés dans une matrice, et de les calculer dans un ordre particulier pour que les résultats nécessaires soient toujours déjà calculés[a]. On pourra utiliser le type `int**` pour représenter une matrice d'entiers, comme dans le TP 5 : une matrice (ici carrée) de taille $m$ est représentée par un tableau de $m$ pointeurs, chacun pointant vers un tableau de $m$ entiers. L'objectif va être de remplir cette matrice `mat` de sorte que `mat[i][j] = binom(i, j)` pour tout $0 \leq j \leq i \leq n$.

★ **Question 5** Écrire une fonction `int** creer_matrice_nulle(int m)` qui crée une matrice carrée $m \times m$ initialisée à zéro (toutes les cases sont égales à 0).

▶ **Question 6** Écrire une fonction `void detruire_matrice(int** mat, int m)` qui libère la mémoire occupée par une matrice carrée $m \times m$.

On part donc d'une matrice carrée $(n+1)\times(n+1)$ initialisée à zéro. On remplit ensuite la diagonale principale avec des 1 (car $\binom{n}{n} = 1$ pour tout $n$) et la première colonne avec des 1 (car $\binom{n}{0} = 1$ pour tout $n$). La partie supérieure droite de la matrice (où $j > i$) n'est pas utilisée. On remplit ensuite le reste de la matrice en utilisant la formule du triangle de Pascal, en parcourant la matrice ligne par ligne puis de gauche à droite.

▶ **Question 7** Justifier que lorsqu'on remplit la matrice dans cet ordre, les valeurs nécessaires pour calculer $\binom{i}{j}$ sont déjà présentes dans lorsqu'on arrive à la case $(i, j)$.

▶ **Question 8** Écrire une fonction `int binom_memo(int n, int k)` qui calcule $\binom{n}{k}$ en utilisant cette méthode. Tester cette fonction en augmentant progressivement la valeur de $n$ jusqu'à $n = 100$.

★ **Question 9** Proposer une fonction `int binom_memo_mieux(int n, int k)` qui utilise un seul tableau (de taille $n + 1$) au lieu d'une matrice pour stocker les valeurs déjà calculées.

★ **Question 10** Proposer la même chose pour l'exemple du cours.

*a.* Dans le Chapitre 14, on appellera ça de la *programmation dynamique ascendante*.

## Exercice 4 – Recherche dichotomique

▶ **Question 1** Écrire une fonction `bool est_dans(int x, int* T, int n)` qui détermine si l'entier $x$ est présent dans le tableau quelconque $T$ de taille $n$ en utilisant une recherche linéaire (on parcourt le tableau du début à la fin).

L'algorithme de recherche dichotomique permet de rechercher un élément dans un tableau **trié** $T$ de taille $n$ en temps logarithmique en $n$. L'idée est la suivante :

- On regarde l'élément au milieu du tableau, c'est-à-dire l'élément d'indice $m = \lfloor \frac{n}{2} \rfloor$.
- Si $T[m] = x$, alors on a trouvé l'élément.
- Si $T[m] < x$, alors on sait que $x$ ne peut pas être dans la première moitié du tableau (car le tableau est trié), donc on recommence la recherche dans la seconde moitié du tableau, c'est-à-dire dans le sous-tableau $T[m+1], T[m+2], \dots, T[n-1]$.
- Si $T[m] > x$, alors on sait que $x$ ne peut pas être dans la seconde moitié du tableau, donc on recommence la recherche dans la première moitié du tableau, c'est-à-dire dans le sous-tableau $T[0], T[1], \dots, T[m-1]$.

Ici, quand on dit « on recommence », on peut le faire de deux façons : soit en utilisant une boucle (version itérative), soit en utilisant la récursion (version récursive). Commençons par la version récursive.

▶ **Question 2** Dans la description informelle de notre algorithme ci-dessus, on n'a pas précis quel était le cas de base. Lequel est-ce ?

▶ **Question 3** Écrire une fonction récursive `bool est_dans_rec(int x, int* T, int n)` qui détermine si l'entier $x$ est présent dans le tableau trié $T$ de taille $n$ en utilisant une recherche dichotomique. *On rappelle que l'on peut utiliser la syntaxe `&T[i]` pour obtenir un pointeur vers le $i$-ième élément du tableau $T$. Ainsi, dans le cas où $T[m] < x$, on peut appeler la fonction récursive sur le sous-tableau $T[m+1], T[m+2], \dots, T[n-1]$ en écrivant `est_dans_rec(x, &T[m+1], n - m - 1)`.*

▶ **Question 4** Exprimer cette fonction de façon itérative. On pourra utiliser une boucle **`while`** et

deux variables pour stocker les indices de début et de fin du sous-tableau dans lequel on est en train de chercher.

**Exercice 5 – ★ Une fonction récursive particulière.**

On considère la fonction suivante :

```c
int mystere(int n) {
    if (n == 1) return 0;
    if (n % 2 == 0) return 1 + mystere(n / 2);
    return 1 + mystere(n + 1);
}
```

▶ **Question 1** Justifier que cette fonction termine pour tout entier naturel $n \geq 1$, c'est-à-dire que l'on ne peut pas appeler cette fonction avec un argument $n \geq 1$ et se retrouver dans une boucle infinie d'appels récursifs.

▶ **Question 2** Déterminer les premiers records de cette fonction, c'est-à-dire les valeurs de $n$ telles que pour tout $1 \leq k < n$, `mystere(k) < mystere(n)`. Conjecturer leur forme générale.

★ **Question 3** Le démontrer.

**Exercice 6**

Débrouillez-vous[a] !

★ **Question 1** Résoudre les problèmes 18, puis 67 du projet Euler `https://projecteuler.net/`.

★ **Question 2** Résoudre le jour 7 de l'Advent of Code 2024 : `https://adventofcode.com/2024/day/7`.

a. Et posez moi des questions si vous en avez quand même.