TD 1

Récursivité (rappels et un peu plus loin)

Exercice 1 (Récursivité)**.**

1. Que calcule la fonction suivante (donnée en pseudo-code et en C) ?

Fonction Toto(n)

si $n = 0$ **alors**
  retourner 1;
sinon
  retourner $n \times$ Toto$(n-1)$;

```
/* Fonction toto en C */

unsigned int toto(unsigned int n){
  if (n == 0) return 1;
  return n * toto(n - 1);
}
```

2. La suite de Fibonacci est définie récursivement par la relation $u_n = u_{n-1} + u_{n-2}$. Cette définition doit bien entendu être complété par une condition d'arrêt, par exemple : $u_1 = u_2 = 1$. Écrire une fonction qui calcule et renvoie le n-ième terme de la suite de Fibonacci ($n \in \mathbb{N}^*$ donné en argument de la fonction).
3. Écrire une fonction récursive qui calcule le pgcd de deux nombres entiers positifs.
4. Que calcule la fonction suivante ?

Fonction Tata(n)

si $n \leq 1$ **alors**
  retourner 0;
sinon
  retourner $1 +$ Tata $\left(\left\lfloor \frac{n}{2} \right\rfloor\right)$;

```
/* Fonction tata en C */

unsigned int tata(unsigned int n){
    if (n <= 1) return 0;
    return 1 + tata(n / 2);
}
```

5. Il n'est parfois pas suffisant d'avoir un bon cas de base, voici un exemple. En C, que vaut `Morris(1, 0)` ?

```
int Morris(int a, int b) {
  if (a == 0) return 1;
  return Morris(a - 1, Morris(a, b));
}
```

Exercice 2 (La fonction 91 de McCarthy)**.**
Les fonctions récursives mêmes simples donnent parfois des résultats difficiles à prévoir. Pour s'en convaincre voici un exemple. Pour $n > 100$ la fonction 91 de McCarthy vaut $n - 10$. Mais pour $n \leq 100$? Tester sur un exemple... pas trop mal choisi, puis prouver le résultat en toute généralité.

Fonction Tata(n)

si $x > 100$ **alors**
  retourner $x - 10$;
sinon
  retourner
  McCarthy(McCarthy$(x + 11)$);

```
int McCarthy(int x)
{
  if (x > 100) return(x - 10);

  return McCarthy(McCarthy(x + 11));
}
```

Exercice 3 (Tours de Hanoï)**.**
On se donne trois piquets, p_1, p_2, p_3 et n disques percés de rayons différents enfilés sur les piquets. On s'autorise une seule opération : Déplacer-disque(p_i, p_j) qui déplace le disque du dessus du piquet p_i vers le dessus du piquet p_j. On s'interdit de poser un disque d sur un disque d' si d est plus grand que d'. On suppose que les disques sont tous rangés sur le premier piquet, p_1, par ordre de grandeur avec le plus grand en dessous. On doit déplacer ces n disques vers le troisième piquet p_3. On cherche un algorithme (en pseudo-code ou en C) pour résoudre le problème pour n quelconque.

L'algorithme consistera en une fonction Déplacer-tour qui prend en entrée l'entier n et trois piquets et procède au déplacement des n disques du dessus du premier piquet vers le troisième piquet à l'aide de Déplacer-disque en utilisant si besoin le piquet intermédiaire. En C on utilisera les prototypes suivants sans détailler le type des piquets, `piquet_t` ni le type des disques.

```
void deplacertour(unsigned int n, piquet_t p1, piquet_t p2, piquet_t p3);
void deplacerdisque(piquet_t p, piquet_t q); /* p --disque--> q */
```

1. Indiquer une succession de déplacements de disques qui aboutisse au résultat pour $n = 2$.
2. En supposant que l'on sache déplacer une tour de $n - 1$ disques du dessus d'un piquet p vers un autre piquet p', comment déplacer n disques ?
3. Écrire l'algorithme en pseudo-code ou en donnant le code de la fonction `deplacertour`.
4. Combien de déplacements de disques fait-on exactement (trouver une forme close en fonction de n) ?
5. Est-ce optimal (le démontrer) ?

Exercice 4 (Le robot cupide)**.**
Toto le robot se trouve à l'entrée Nord-Ouest d'un damier rectangulaire de $N \times M$ cases. Il doit sortir par la sortie Sud-Est en descendant vers le Sud et en allant vers l'Est. Il a le choix à chaque pas (un pas = une case) entre : descendre verticalement ; aller en diagonale ; ou se déplacer horizontalement vers l'Est. Il y a un sac d'or sur chaque case, dont la valeur est lisible depuis la position initiale de *Toto*. Le but de *Toto* est de ramasser le plus d'or possible durant son trajet.

On veut écrire en pseudo-code ou en C, un algorithme Robot-cupide(x, y) qui, étant donnés le damier et les coordonnées x et y d'une case, rend la quantité maximum d'or (gain) que peut ramasser le robot en se déplaçant du coin Nord-Ouest jusqu'à cette case. En C, on pourra considérer que le damier est un tableau bidimensionnel déclaré globalement et dont les dimensions sont connues.

A	B
C	D

1. Considérons quatre cases du damier comme ci-dessus et supposons que l'on connaisse le gain maximum du robot pour les cases A, B et C, quel sera le gain maximum pour la case D ?
2. Écrire l'algorithme.
3. Si le robot se déplace d'un coin à l'autre d'un damier carré 4×4 combien de fois l'algorithme calcule t-il le gain maximum sur la deuxième case de la diagonale ? Plus généralement, lors du calcul du gain maximum sur la case x, y combien y a-t'il d'appels au calcul du gain maximum d'une case i, j ($i \leq x$, $j \leq y$).