TD/TP n°4 (IF122)
Programmation de fonctions récursives

Le jeu des allumettes (Nim ou jeu de Marienbad)

Le jeu se joue à deux. Plusieurs allumettes (représentée par des 1) sont disposées sur 21 emplacement.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1			1	1	1	1				1	1				1	1		1	1	1

A chaque tour vous pouvez retirer une, deux ou trois allumettes se situant dans des cases consécutives. Le but du jeu est de faire retirer la dernière allumette par son adversaire. Autrement dit, celui qui ramasse la dernière allumette perd.

Toute position du jeu peut être représentée par la structure suivante :

```
typedef struct position {
  int damier [21];     // damier[0..20], 0 => vide, 1 => allumette
  int player;          // 0 => ordi, 1 => humain
};
```

Par ailleurs, le coup d'un joueur peut être représenté par la structure suivante :

```
typedef struct coup {
  int pos;            // 0..20 position de la première case concernée
  int nb;             // 1..3  nombre d'allumettes retirées
};
```

Par exemple, le coup par lequel on enlève les deux allumettes des cases 10 et 11 de la position ci-dessus, sera codé par une variable `c` de type `coup` telle que `c.pos=10` et `c.nb=2`.

Exercice 1

Écrire une fonction **récursive** `bool coup_possible(position p, coup c)` qui vérifie si le coup c est possible dans la position p donnée (il faut qu'il y ait assez d'allumettes à enlever que le coup le demande). En ce qui concerne le caractère récursif du problème, pensez que vérifier la possibilité d'enlever `n` allumettes voisines, signifie vérifier de pouvoir en enlever une, puis de pouvoir enlever les `n-1` voisines...

Exercice 2

Écrire une fonction **récursive** `position mouvement (position p, coup c)` qui retourne la nouvelle position du jeu, après le coup c dans la position p (on peut supposer que le coup soit possible). Il s'agit, en d'autres termes, d'effacer des allumettes (mettre à 0 des cases du damier). À nouveau, si on doit en effacer `n`, on peut toujours commencer par en effacer une, pour ensuite effacer les `n-1` suivantes... Suggestion : ne pas oublier aussi de changer de main (les joueurs s'alternent).

Exercice 3

Écrire une fonction **récursive** `coup entrer_coup (position p)` qui demande à l'utilisateur d'entrer les deux valeurs définissant son coup `(pos,nb)` *tant que* le coup n'est pas un coup *possible* dans la position `p` donnée.

Exercice 4

Une position p est *terminale* lorsque il n'y a plus d'allumettes. Autrement dit, lorsque toutes les valeurs du damier `p.damier[]` sont à zéro. La fonction suivante permet de réaliser ce test :

```
bool terminale (position p) {
int i = 0 ;
while ((i<=20) && (p.damier[i]==0)) i++ ;
return (i>20) ;
}
```

Écrire une version équivalente de la fonction ci-dessus s'appuyant sur un procédé **récursif** et non pas sur une boucle `while`.

Exercice 5

Il est possible de déterminer si une position p du jeu est *gagnante* pour le joueur qui a la main (`p.player`), par un raisonnement **récursif**. En effet, si la position est terminale le joueur en question a gagné, puisque au tour précédent, son adversaire a enlevé la dernière allumette. En revanche, si la position n'est pas terminale, alors il suffit d'explorer tout les coups possibles : si on trouve un coup conduisant à une position `p'` perdante (non *gagnante*...) pour son adversaire, la position `p` est *gagnante*, sinon elle est perdante. Suivant cette idée, écrire une fonction récursive :

```
bool gagnante (position p)
```

Exercice 6

Supposer à présent d'avoir à disposition :

- une procédure d'affichage d'une position : `void affiche(position p)` (très simple à écrire en TP), produisant une sortie de ce genre :

```
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
| 0| 1| 2| 3| 4| 5| 6| 7| 8| 9|10|11|12|13|14|15|16|17|18|19|20|
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
| 0| 1| 1| 1| 0| 1| 1| 0| 1| 1| 0| 0| 1| 0| 0| 0| 0| 1| 1| 0| 1|
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
Player : 0
```

- une fonction qui calcule le meilleur coup pour l'ordinateur : `coup ordi_malin (position p)` (très simple à écrire en TP, en utilisant `gagnante()`)

Écrire alors une procédure **récursive** :

```
void boucle_jeu(position p)
```

qui réalise la boucle d'interaction du jeu, c'est-à-dire la boucle où la position courante est affichée et l'utilisateur est sollicité pour entrer son coup, lorsque c'est à lui de jouer, ou bien la fonction `ordi_malin()` est appelée, lorsque c'est au tour de la machine. Cela jusqu'à la fin du match.