Tours de Hanoï (fonctions récursives, tableaux, niveau 3)

> Exercice n°35 (pages 85 et 257) de l'ouvrage *C++ par la pratique*.

Description du problème

On dispose d'un plateau de jeu de tour de Hanoï constitué de 3 piliers (cf le cours ICC, leçon I.2).

N disques de diamètres décroissants sont placés sur le premier pilier, Les deux autres étant vides.

Exemple avec N=4 :

L'objectif est de déplacer les N disques du premier au dernier pilier en utilisant si nécessaire le pilier du centre comme pilier de stockage.

On ne peut déplacer **qu'un seul disque à la fois**, et il faut respecter à tout moment la **contrainte** suivante :
aucun disque ne peut être posé sur un disque plus petit que lui.

Soit `void hanoi(n, A, B)` la fonction qui déplace les `n` disques du dessus du pilier `A` vers le pilier `B` (`A` et `B` sont des chiffres entre 0 et 2).

Cette fonction peut être écrite de façon récursive avec les constatations suivantes :

1. si `n=0`, il n'y a rien à faire.
2. si l'on sait résoudre le problème pour `n-1` disques, on sait le résoudre pour `n` disques (`n > 0`).
   a. déplacer les `n-1` premiers disques du pilier `A` au pilier `C` (avec `C != A` et `C != B`).
   b. déplacer le $n^{ième}$ disque de `A` à `B`.
   c. déplacer les `n-1` disques de `C` à `B`.

(voir le cours ICC I.2 pour plus de détails si nécessaire)

À faire

Il faut commencer par définir les structures de données qui vont être utilisées pour représenter le jeu.
On va pour cela représenter les disques par leur rayon : définissez par exemple

- le type `Disque` comme étant un entier non signé (rayon du disque) ;
- la constante `PAS_DE_DISQUE` comme étant le `Disque` de rayon nul (c.-à-d. donc simplement la constante `0` ;
- la constante `N` représentant la taille initiale de la tour à déplacer
  (les plus motivés pourront rendre ce paramètre variable et le demander à l'utilisateur).

Le jeu sera alors représenté comme un tableau de 3 piliers, chaque pilier étant un tableau dynamique de `Disque`s.

Définissez ensuite 3 fonctions principales :

1. une fonction , par exemple `void init(Jeu& jeu);`, qui initialise le jeu
   (ceux, les plus motivés, qui ont rendu la taille du jeu variable devront bien sûr ici en tenir compte ici) ;
2. une fonction qui affiche le jeu , par exemple `void affiche(const Jeu& jeu);` ;
3. et une fonction , par exemple `hanoi` (prototype donné plus tard ci-dessous), qui résout (récursivement) le problème du déplacement d'une tour.

Le corps principal du programme se limitant alors à :

```
int main()
{
  Jeu monjeu;

  init(monjeu);
  affiche(monjeu);
  hanoi(N, 0, 2, monjeu);
```

```
    return 0;
}
```

La fonction d'initialisation met simplement un disque de rayon `i` à la `i`-ème position du premier pilier, et `PAS_DE_DISQUE` à toutes les positions des autres piliers.

Pour la fonction qui affiche l'état du jeu (voir l'**exemple de déroulement** ci-dessous), je vous propose d'adopter une approche modulaire (décomposition de la tâche en sous-tâches) :

a. la fonction qui affiche le jeu complet utilise une autre fonction , par exemple `void affiche(const Disque& d);`, pour afficher un disque ;
b. cette autre fonction utilise une troisième fonction , par exemple `void affiche(char c, unsigned int n = 1);`, affichant `n` caractères identiques consécutivement.

Écrivez enfin la fonction

```
void hanoi(unsigned int n, unsigned int origine, unsigned int destination,
    Jeu& jeu)
```

de sorte qu'elle déplace `n` disques de la tour `origine` au pilier `destination`, en utilisant les principes présentés plus haut.

On utilisera également une approche modulaire pour la fonction `hanoi` :

- définir une fonction

  ```
  void deplace(Pilier& origine, Pilier& destination)
  ```

  qui, si le déplacement est valide, déplace le premier disque du pilier `origine` au pilier `destination`.

- définir une fonction

  ```
  unsigned int autre(unsigned int p1, unsigned int p2)
  ```

  qui, étant donné les numéros de deux piliers (`p1` et `p2`), détermine le numéro du troisième pilier.
  Par exemple `autre(0, 2)` vaut `1`, `autre(2, 1)` vaut `0`, etc.

  Indice : utilisez la somme `p1+p2`, sachant que la sommes des 3 numéros de piliers fait toujours 3.

Exemple de déroulement

Voici un exemple pour N=3 :

```
  -      |      |
 ---     |      |
-----    |      |
#################

  |      |      |
 ---     |      |
-----    |      -
#################

  |      |      |
  |      |      |
-----   ---     -
#################

  |      |      |
  |      -      |
-----   ---     |
#################

  |      |      |
  |      -      |
  |     ---   -----
#################

  |      |      |
  |      |      |
  -     ---   -----
```