TDM3 **Compilation séparée, Tours de Hanoi**

---

1 La programmation modulaire

Le découpage en plusieurs modules a plusieurs avantages : lisibilité, aide au débogage, possibilité de réutiliser facilement une partie du code pour d'autres applications, possibilité de masquer l'implémentation d'un TAD, possibilité de ne recompiler que les fichiers ayant été modifiés.
Les fichiers créés lors du découpage sont de deux types :
– fichier en-tête ou header (.h)
– fichier source (.c)
Une règle d'écriture est d'associer à chaque fichier source *nom.c* un fichier en-tête *nom.h* sauf pour le fichier contenant la fonction *main* (souvent nommé *main.c*) .
Le fichier *nom.h* contient les déclarations des fonctions non locales au fichier *nom.c*, les définitions des constantes symboliques et des types partagés par les deux fichiers ainsi que diverses directives au préprocesseur.
Le fichier *nom.c* associé se compose des définitions des fonctions dont la déclaration se trouve dans le fichier *nom.h* ainsi que d'éventuelles fonctions locales à *nom.c* et de directives au pré-processeur.
Le fichier *nom.h* doit être inclus dans le fichier *nom.c* et dans tous les autres fichiers utilisant des objets déclarés dans *nom.h*. On se prémunit du risque d'inclusions multiples d'un fichier en-tête en créant pour chaque fichier *nom.h* une constante symbolique, habituellement notée : NOM_H, au début du fichier *nom.h*. Si cette constante est déjà définie, c'est que le fichier *nom.h* a déjà été inclus et le compilateur ne le reprend pas en compte. Sinon, la constante est définie et on prend en compte le contenu de *nom.h*.
La compilation peut se faire directement en ligne de commande ou bien être automatisée par l'utilisation d'un fichier makefile (voir document joint).

1.1 Mise en oeuvre, l'exemple d'une pile statique avec masquage :

On dispose du fichier *PileStatiqueFichierDeDepartPourCompilationSeparee.c* Ce fichier a été préparé afin que le masquage de l'implémentation de la pile puisse avoir lieu lors du découpage en module. Le principe du masquage est de définir un objet comme pointeur sur un type masqué ; ici le type T_pile (destiné à être visible dans le fichier pile.h) est défini comme pointeur sur la structure struct cell (cachée dans pile.c)
A partir du mono-fichier, on a crée les 5 fichiers :
· elt_pile.h
· elt_pile.c

· pile.h
· pile.c
· main.c

Si besoin, par la suite, on pourra créer une pile d'objets d'autres types (réels,chaînes de caractères,structures...) en adaptant les définitions des fonctions de elt_pile.c

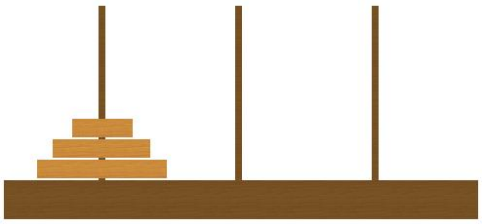
◇ Exercice **1** : Testez le bon fonctionnement de la pile en compilant d'abord en ligne de commande puis en utilisant le makefile fourni.
Dans la fonction *initialiser_pile*, remarquez l'allocation dans le tas de la structure. En quoi le masquage rend-il nécessaire cette allocation ?

◇ Exercice **2** : Effectuer le même travail de découpage en 5 fichiers pour votre pile dynamique. Si vous procédez avec méthode, **le seul fichier à modifier** est le fichier *pile.c*

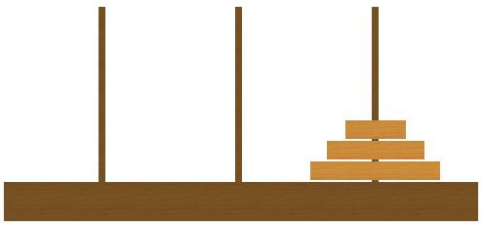
2 Les tours de Hanoi

Le jeu des tours de Hanoi, est un casse tête composé de 3 tours numérotées 1, 2, 3 et de N disques (ou anneaux) perforés de différentes tailles. Le but de ce jeu, est de passer d'une configuration initiale (tous les disques sur la tour 1) à une configuration finale (tous les disques sur la tour 3).
On peut enlever l'anneau supérieur d'une tour pour le placer sur une autre tour à condition que à tout moment il n'y ait jamais un anneau recouvert par un anneau plus grand.
Configurations avec N=3 anneaux :

Configuration initiale :



Configuration finale :



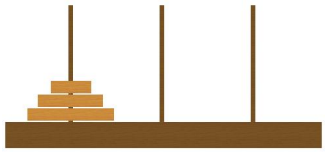
On souhaite écrire un programme C non récursif qui utilise une pile pour afficher à l'écran les différents déplacements d'anneaux nécessaires pour résoudre le problème avec N disques.

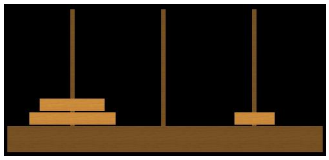
• Analyse du problème :
Si on observe le jeu de plus prés (pour plus de 1 disque) on s'aperçoit que quelque soit la position, il n'y a que 1 ou 2 déplacements possibles.
Le petit disque peut toujours se déplacer sur les deux autres piques, et si un disque différent

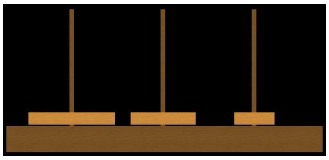
du plus petit peut être déplacé, il n'a qu'une seule possibilité (sur la pique où n'est pas le petit disque).
Un algorithme itératif est le suivant :

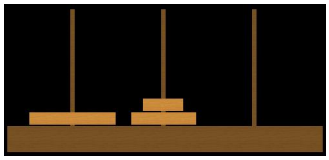
```
début   initialiser ;
        Tant que (les piques 1 et 2 ne sont pas toutes deux vides) faire
            début Déplacer le petit disque dans le sens antitrigo (132132...)
                Si on peut déplacer un autre disque que le petit
                alors le déplacer où l'on peut
            fin
fin
```

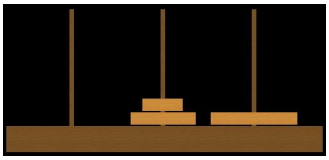
• Exemple de résolution avec N=3 disques :

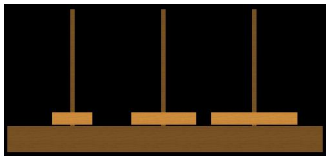
Départ : 

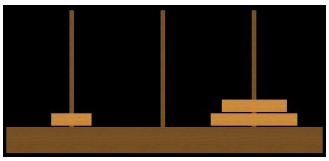
Etape 1 : 

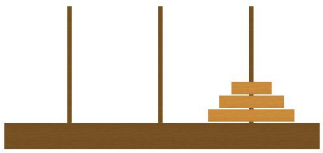
Etape 2 : 

Etape 3 : 

Etape 4 : 

Etape 5 : 

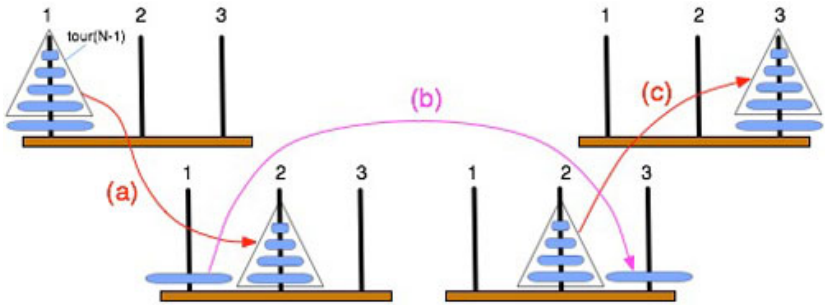
Etape 6 : 

Etape 7 : 

◇ Exercice **3** : Ecrire un main (dans un fichier *hanoi.c* par exemple) qui affiche la liste des déplacements d'anneaux à effectuer pour résoudre le problème des tours de Hanoi. On affichera aussi le nombre de déplacements nécessaires.
On utilisera les fichiers *element_pile.h*, *element_pile.c*, *pile.h*, *pile.c* précédents. Le programme devra pouvoir fonctionner sans modification avec la pile statique ou une pile dynamique.

Pour aller plus loin :

• Résolution récursive du problème des tours de Hanoi :
Ecrire une procédure récursive **hanoi(int N, int tour_depart, int tour_fin, int tour_aux)** en s'appuyant sur le schéma suivant :



Compter le nombre de déplacements d'anneaux et comparer avec la version itérative.

- Améliorer l'algorithme itératif en tenant compte de la parité du nombre de disques afin de diminuer le nombre de déplacements.

◇ ◇ ◇◇ ◇ ◇◇