# TP noté de programmation impérative avancée

**ÉNSIIE, semestre 2**

mercredi 16 mars 2016

Durée : 1h45.
1 page A4 recto verso autorisée.
Les diapositives du cours sont disponibles dans `/pub/sujets/`.

Le problème des tours de Hanoï est le suivant : on dispose de trois piques (numérotées 1, 2 et 3) sur lesquelles peuvent être insérées des disques de diamètres différents. On impose comme contrainte qu'un disque ne peut être posé sur une pique que sur un disque de diamètre plus grand. Initialement, $n$ disques sont rangés sur une pique (pique d'origine), dans l'ordre décroissant de diamètre par conséquent. L'objectif est de déplacer tous les disques sur une deuxième pique (pique de destination), en utilisant éventuellement la troisième (pique auxiliaire) mais surtout en respectant tout le temps la contrainte énoncée précédemment.

L'objectif de ce TP est d'implémenter l'algorithme de résolution du problème des tours de Hanoï en utilisant une file pour stocker les mouvements à réaliser. On rappelle qu'une file est une structure de donnée dans laquelle on peut enfiler des éléments, et dans laquelle les éléments sont récupérés selon le principe "premier inséré, premier sorti". Ici, les éléments seront des mouvements, c'est-à-dire des paires d'entiers indiquant une pique d'origine et une pique de destination.

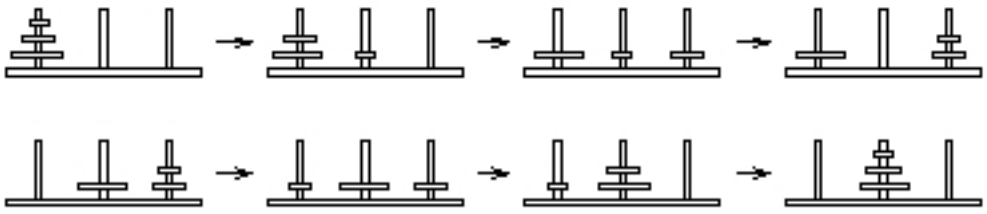

FIGURE 1 – Résolution du problème des tours de Hanoï pour $n = 3$

Le problème des tours de Hanoï peut être résolu très facilement à l'aide de l'algorithme récursif suivant. `hanoi(n, a, b, c)` prend quatre arguments : le nombre de disques $n$, le numéro de la pique d'origine $a$, le numéro de la pique de destination $b$ et le numéro de la pique auxiliaire $c$.

— Si $n = 1$, alors on bouge le disque de la pique $a$ à la pique $b$.
— Sinon,
  — on déplace récursivement $n - 1$ disques de la pique $a$ à la pique $c$, en utilisant $b$ comme pique auxiliaire. (`hanoi(n-1, a, c, b)`)
  — on bouge le dernier disque de la pique $a$ à la pique $b$ ;
  — on déplace récursivement $n - 1$ disques de la pique $c$ à la pique de $b$, en utilisant la pique $a$ comme pique auxiliaire (`hanoi(n-1, c, b, a)`).

Il ne vous est pas demandé de montrer que cet algorithme est correct.

Il vous est fourni (dans `/pub/sujets/`)

— un module `mouvement` (`mouvement.c` et `mouvement.h`), qui contient
  — un type abstrait pour les mouvements ;
  — une fonction `constr` permettant de construire un mouvement à partir de deux entiers ;
  — une fonction `orig` qui donne l'entier origine d'un mouvement ;
  — une fonction `dest` qui donne l'entier destination d'un mouvement.
— l'implémentation d'un module de file (`file_liste.c`) à l'aide de listes.

Il vous faudra écrire (les détails vous sont donnés dans la suite de l'énoncé) :

— l'interface du module de file (`file.h`) ;
— une implémentation des files à l'aide de tableaux circulaires (`file_tab.c`) ;
— un module sans interface `hanoi.c` qui contient des fonctions pour résoudre le problème des tours de Hanoï, pour afficher une file de mouvements, et une fonction principale ;
— un `Makefile` permettant de créer des exécutables utilisant chacun une des implémentation des files.

Une fois l'exercice 1 réalisé, les trois autres sont indépendants.

## Exercice 1 : Interface des files

Dans un fichier `file.h`, écrire l'interface du module des files contenant des mouvements qui devra comprendre les déclarations suivantes :

1. un type abstrait `file` pour les files ;
2. une fonction `file_vide` sans argument qui crée une file vide ;
3. une fonction `est_file_vide` qui prend une file et qui teste si elle est vide ;
4. une fonction `enfiler` qui prend en argument une file par référence et un mouvement, et qui insère le mouvement dans la file par effet ;
5. une fonction `defiler` qui prend une file par référence et qui retourne le mouvement en tête de la file après l'avoir retiré de la file par effet ; le comportement est non défini si la pile est vide.

## Exercice 2 : Tours de Hanoï

Dans un fichier `hanoi.c` écrire les fonctions :

— `hanoi` qui attend cinq arguments : le nombre de disques $n$, les numéros de la pique d'origine $a$, de destination $b$ et auxiliaire $c$, et en plus une file par référence $*f$ dans laquelle on stockera les mouvements des disques ; on utilisera l'algorithme pour résoudre le problème des tours de Hanoï décrit ci-dessus.

— `affiche_file` qui prend en argument une file (par valeur) et qui affiche successivement sur la sortie standard `Mouvement de la pique` $i$ `vers la pique` $j$ pour tous les mouvements $(i, j)$ contenus dans la file.

— `main` qui convertit en entier le premier argument passé à l'exécutable en ligne de commande, qui appelle `hanoi` sur cet entier avec une file vide, puis qui affiche la file.

## Exercice 3 : Makefile

Écrire un `Makefile` avec toutes les dépendances nécessaires pour construire deux exécutables `hanoi_liste` et `hanoi_tab` qui utilisent chacun l'implémentation des files adéquate.

## Exercice 4 : Implémentation à l'aide de tableaux

Pour implémenter une file, on utilise un tableau avec deux indices `debut` et `fin` qui indiquent les cases du tableau au début et juste après la fin de la file. Par exemple
`[ (0,39); (-3,23); (42,0); (1,1); (2,2); (3,3); (4,4); (26,42); (-13,-2) ]`
avec `debut = 3` et `fin = 7` représente une file contenant les éléments de `(1,1)` à `(4,4)` insérés dans l'ordre croissant. Quand on dépasse la fin du tableau on revient au début. Par exemple le tableau
`[ (3,3); (4,4); (42,-23); (14,0); (-13,-3); (7,7); (-3,-24); (1,1); (2,2) ]`
avec `debut = 7` et `fin = 2` représente aussi une file contenant les éléments de `(1,1)` à `(4,4)` insérés dans l'ordre croissant. Si on veut insérer plus d'éléments dans la file qu'il n'y a de cases, on redimensionne dynamiquement le tableau.

— Pour créer la file vide, on renvoie un tableau de taille 3 avec des indices de début et de fin tous deux égaux à 1.

— Pour savoir si la file est vide ou pas, on regarde si `debut` est égal à `fin`.

— Pour défiler une file non vide, on incrémente `debut` de 1, en le mettant à 0 s'il dépasse la taille du tableau. On retourne l'élément de la case située à l'ancienne valeur de `debut`.

— Pour insérer un élément, on incrémente `fin` de 1 (en le mettant à 0 s'il dépasse la fin de la liste). Si le nouvel indice de fin n'est pas égal à celui de début (auquel cas la file est pleine, on doit redimensionner), on insère le nouvel élément dans la case indiquée par l'ancien indice de fin.

  Pour redimensionner, on alloue un nouveau tableau de taille double, on insère tous les éléments de l'ancien tableau entre `debut` et `fin` dans le nouveau tableau, en partant de la case 0 du nouveau tableau, puis l'élément à insérer. On fixe `debut` à 0 et `fin` au nombre d'éléments insérés. On n'oubliera pas de désallouer l'ancien tableau.

Implémenter les files à l'aide de tableaux circulaires dans le fichier `file_tab.c`.