CRO TP1 :

Pointeurs et Structures : Les Tours de Hanoï

(1 séance sur machine), 13 juillet 2023

But du TP

Le but de ce TP est de vous familiariser avec la manipulation des pointeurs dans le langage C. Ce TP introduit un exemple de *gestion dynamique* de la mémoire. La *décomposition modulaire* d'un projet logiciel est aussi illustrée ici, ainsi que l'écriture d'un Makefile complet.

1 Retour sur la fin du TD4

Pour ceux qui n'ont pas eu le temps de finir le TD sur les pointeurs, nous repartons sur la fin du TD qui introduit une notion importante : les pointeurs associées aux structures en C

Retour sur les structures en C

Déclaration de structure : Voici comment une variable `etudiant1` qui contient trois champs d'information ainsi qu'un champ `suivant` qui permet de faire une liste d'étudiants. On donne un *label* à la structure (ici `student`) qui est un raccourcit pour la nommer. Notez que la dernière instruction ci-dessous (`typedef`) permet de définir un nouveau type `ETUDIANT` qui est un raccourçit pour `struct student`.

```
struct student        //declaration de la structure
{
  char nom[100];
  char prénom[100];
  int numero_etudiant;
  struct student *suivant;
};
struct student etudiant1;   //declaration de la variable etudiant1
typdef struct student ETUDIANT;   // déclaration du type ETUDIANT
```

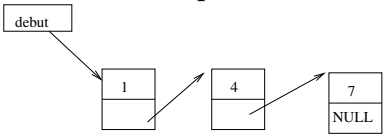
Utilisation d'une variable de type de structure :

```
strcpy(etudiant1.nom,"Stanica");
strcpy(etudiant1.nom,"Razvan");
etudiant1.numero_etudiant=4328907394;
etudiant1.suivant=NULL;
```

En C, on associe souvent les structures et les pointeurs pour faire des **listes chaînées**.

Structures auto-référencées

On a souvent besoin en C de modèles de structure dont un des membres est un *pointeur vers une structure de même modèle*. Cette représentation permet en particulier de construire des listes chaînées. Un élément de la liste chaînée est une structure appelée qui contient la valeur de l'élément et un pointeur sur l'élément suivant. Le dernier élément pointe sur la liste vide `NULL`.



C'est la façon dont on implémente le type `list` en C. Une liste est une collection d'élément de même type, mais contrairement aux tableaux, cette structure est dynamique, on peut rajouter ou enlever des élément au cours de l'exécution du programme. La contrepartie est qu'on ne peut pas accéder directement à chaque élément de la liste. En général on accède au premier élément, à partir duquel on peut *parcourir* la liste pour trouver l'élément recherché. Ce mécanisme est implémenté par les liste chaînées constituées de l'association de structure et de pointeurs.

QUESTION 1 ▶ On considère le type `ELEMLIST` suivant qui définit un élément d'une liste d'entier, ainsi que le type `LIST` qui désigne une liste d'entier :

```
struct model_elem
{
  int val;
  struct model_elem *suivant;
};
typedef struct model_elem ELEMLIST;
typedef ELEMLIST *LIST;
```

Comprenez vous la définition ci-dessus ?
Avez vous compris qu'une liste (type `LIST`) c'est la même chose qu'un pointeur vers un `ELEMLIST` (type `ELEMLIST *`) ?
Demander à l'enseignant si vous avez un doute.
Entrez le code ci-dessus dans un fichier `list.h`.

QUESTION 2 ▶ Créez un fichier `list.c` avec une fonction `main()`. Ecrire une une fonction :
`ELEMLIST *new_list(int val);`
qui crée un nouvel élément de liste contenant la valeur `val` et retourne un pointeur vers cet élément. ATTENTION, la fonction devra utiliser un `malloc` pour allouer le nouvel élément et utilisera une variable locale qui est bien *un pointeur* sur un `ELEMLIST`.

QUESTION 3 ▶ Appelez votre fonction `new_list()` dans la fonction `main()` et testez votre programme en le compilant :
`gcc -Wall main.c -o main`

QUESTION 4 ▶ Écrire le code C d'une fonction :
`LISTE ajouterListe (LISTE liste1, int val)`
qui insère un nouvel entier dans la liste **en tête de liste** et retourne la nouvelle liste
Maintenant que nous avons eu un premier contact avec les liste en C, nous allons les utiliser pour résoudre un problème classique : les tours de hanoï.

2 Description du problème des tours de Hanoï

Il s'agit de programmer la résolution du problème des tours de Hanoï pour des tours de taille N (N n'étant pas forcément connu à la compilation). Les tours de Hanoï seront modélisées grâce à une structure de pile. La décomposition de programme en modules ainsi que les prototypes des fonctions manipulant les piles vous sont proposées en téléchargement (voir section **??**).

On rappelle le principe des tours de Hanoï déjà vu cours ALG. Les tours de Hanoï sont 3 pics sur lesquels sont insérées N rondelles de tailles croissantes. Au départ les rondelles sont sur le pic de gauche. Le but est de toutes les déplacer sur le pic de droite. Par exemple, Les positions des rondelles au départ et à la fin, pour des tours de hanoï de taille 4 sont illustrées en figure 1.

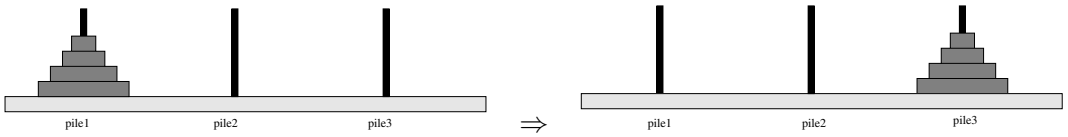


FIGURE 1 – résolution du problème des tours de Hanoï de taille 4

Une étape correspond au déplacement d'une rondelle, sachant que :

— On ne peut déplacer que les rondelles situées au sommet des tas.

— On ne peut poser une rondelle que sur une rondelle de plus gros diamètre (ou sur le socle).

On modélisera le problème en utilisant trois piles d'entiers pour les trois pics. Une rondelle est donc représentée par un entier (correspondant à la largeur de la rondelle). Le résultat du TP sera une fonction `hanoi` permettant de résoudre le problème de Hanoï en utilisant uniquement la fonction `déplacer` qui déplace une rondelle d'un pic à un autre.
On rappelle qu'une pile est une structure dans laquelle les ajouts et les retraits ont lieu au sommet de pile : l'élément dépilé est le dernier à avoir été empilé. En C, une pile est implémentée par une liste chaînée pour laquelle l'insertion et le retrait d'élément aura lieu *en tête de liste*. Pour ce TP, on utilisera le type suivant pour une pile :

```
/* Element de pile */
struct model_elem {
  int elem ;
  struct model_elem* suivant;
};
typedef struct model_elem ELEMPILE;

/* Pile */
typedef ELEMPILE *PILE;
```

3 Ecriture des fonctions de manipulation de pile

Un squelette vous est fourni (fichier `TP1-init.tar`) sur Moodle. Cette archive contient les fichiers `pile_type.h`, `pile.h` et `hanoi.h` proposant une décomposition modulaire en trois fichier (`pile.c`, `hanoi.c` et `main.c`. Le travail du TP consistera essentiellement en l'écriture dans le fichier `pile.c` des fonctions dont le prototype est décrit dans le fichier `pile.h`, ainqi que l'écriture du fichier `hanoi.c`. Le fichier `main.c` présent dans l'archive sera utilisé pour tester les fonctions de manipulation de pile **au fur et à mesure de leur écriture**.
Pour extraire l'archive on peut exécuter la commande suivante : `tar -xvf TP1-init.tar`
Cela créé un repertoire `code`, dans lequel vous pouvez travailler.

QUESTION 5 ▶

1. Récupérez l'archive sur Moodle et créer le Makefile (pour cela créer les fichiers `pile.c` et `hanoi.c`). **faites valider par l'enseignant**. Si vous n'avez pas fait de Makefile, vous ne devez pas continuer le TP. Il est normal que la commande `make` échoue car les fonctions `Empiler`, `Depiler`, etc. ne sont pas définies.
2. Programmez les fonctions de manipulation d'une pile dans le fichier `pile.c` :
   (a) D'abord regarder le fichier `pile.h` pour comprendre ce que doit faire chaque fonction. Programmez la fonction `error1` et vérifiez qu'elle fonctionne avec votre `main`.
   (b) Commencez par la fonction `afficherPile` sans laquelle on ne peux pas vérifier que nos piles sont correctes.
   (c) Pogrammez ensuite la fonction `Empiler`, vérifiez son fonctionnement.

(d) Enfin la fonction `Depiler`, attention vous remarquerez que la fonction `Depiler` prend en argument un pointeur sur une pile (donc un pointeur sur un pointeur sur un élément). La raison en est, bien évidement, qu'une fonction C ne peux pas modifier un argument. Cette fonction renvoyant déjà l'éléménet dépilé, elle ne peut en même temps renvoyer la pile modifiée. Ce mécanisme est utilisé **systématiquement dans le langage C** : lorsqu'une fonction modifie un de ses argument, on passe en argument un pointeur sur l'argument plutot que l'argument lui-même. Pour bien comprendre cela, on rappelle le principe des passages d'arguments par valeur en C, si vous aves encore des doutes avec le `PILE* pile`, demandez à un enseignant. Enfin, n'oubliez pas, lorsque vous dépilez, de libérer la mémoire avec la fonction `free`

passage de paramètre en C

Le mécanisme de passage de paramêtre en C (on dit : passage de paramêtre par valeur) doit être bien compris :

Passage de paramêtre par valeur

— Ce qui est transmis à la fonction est *une copie de la valeur de l'argument*

— dans `int factorielle(int n)`, `n` est le *paramètre formel* de la fonction. Il peut être utilisé dans le corps de la fonction comme une variable locale.

— dans `x=factorielle(10);`, `10` est le *paramètre effectif* utilisé lors de cet appel.

— En C, tout se passe lors de cette appel comme si on exécutait le corps de la fonction *avec la case mémoire pour* `n` *contenant une copie de la case mémoire contenant 10*. On dit que les paramètres sont passés *par valeur*.

— Lorsque l'appel est terminé, la case mémoire de `n` disparait.

— Donc : on peut modifier la valeur du paramètre formel `n` dans le corps de la fonction mais cela ne modifiera pas la valeur du paramètre effectif (10).

Passage de paramêtre par référence

— La seule manière, pour une fonction C, de modifier son argument et de travailler sur *un pointeur sur l'argument*. On parle alors de *passage par référence* (une *référence* est un autre nom pour un pointeur). C'est un abus de langage, le passage est toujours par valeur, mais on passe une référence sur l'objet.

— Ce point particulier (passage par valeur ou par référence) est un des point qui diffèrencie les différents langages de programmation, il est important de comprendre ce que cela implique, ainsi que de comprendre comment cela est implémenté (juste par un pointeur... ou par une copie).

4 Résolution du problême des tours de hanoï

QUESTION 6 ▶ Programmer la résolution du problème des tours de Hanoï pour des tours de taille N, pour cela on utilisera les prototypes de fonctions donnés dans le fichier `hanoi.h`. Testez votre programme, mesurez les performances pour des valeurs croissantes de N.

QUESTION 7 ▶ Lorsque le programme sera réalisé, écrivez une fonction d'affichage de pile dédiée à ce TP qui affiche convivialement les tours à l'écran.