Réponses Questions TP5 :

I) Les variables dévoilent pour nous leurs adresses – L'opérateur &

1. Qu'observe-t-on lors de l'exécution de ce programme ?

Nous pouvons voir que ce programme affiche pour chaque case du tableau, son indice, sa valeur, et l'adresse de la variable en mémoire.
Nous remarquons que le tableau est créé avec des adresses mémoires qui ne sont pas à la suite. En effet, le tableau stocke les variables a un intervalle de 4 adresses mémoires. (Il saute de 4 en 4).

2. Illustrez sous forme d'un schéma (plan mémoire) similaire à celui de la Figure 1 la représentation en mémoire du tableau « t ».

Adresses (&t[0]=)	Mémoire (Variable t[0]=)
1000	10
...	...
1004	20
...	...
1008	30
...	...
1012	40
...	...
1016	50

3. Compilez et exécutez plusieurs fois ce programme en remplaçant dans la déclaration du tableau « t » le type « int » par « short int », « float », « double ». Que peut-on en déduire sur le format de stockage des types de données mentionnés ?

L'écart entre deux adresses successives dépend de la taille du type de données. Avec un short int, il y a un pas de 2; avec un float, un pas de 4 et avec un double, un pas de 8.

4. Écrire le programme suivant et expliquer à l'aide d'un schéma son fonctionnement et sa représentation mémoire.

Le programme alloue dynamiquement de la mémoire pour un entier avec malloc(), et il vérifie que px n'est pas NULL. Puis il stocke la valeur 1 dans cette mémoire, et il affiche l'adresse du pointeur px, l'adresse allouée et la valeur stockée. A la fin, la mémoire est libérée avec free().

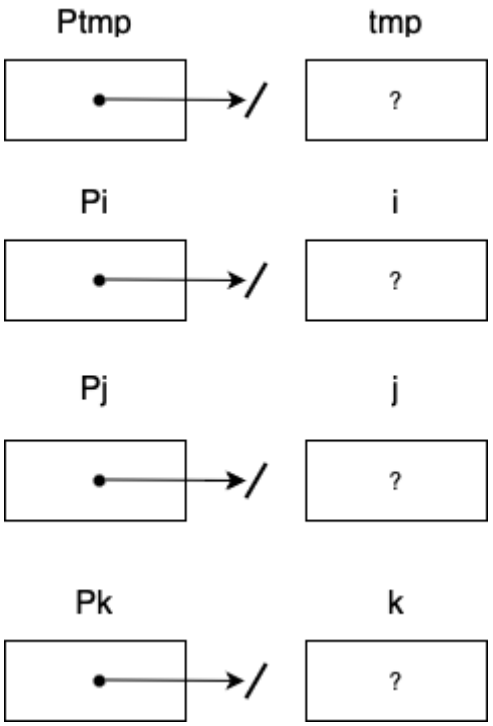
Adresse	Nom	Valeur
1000	px	2000
2000	*px	1

- px (stocké en 1000) contient 2000, l'adresse allouée par malloc().
- *px (stocké en 2000) contient la valeur 1.

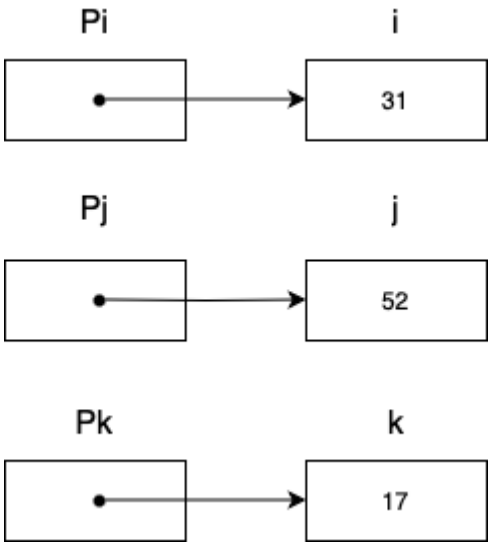
II) Accéder au contenu des variables de manière indirecte: L'opérateur *

Le code fait trois passages sur les variables i,j,k et leurs pointeurs. Nous avons schématisé ces passages.

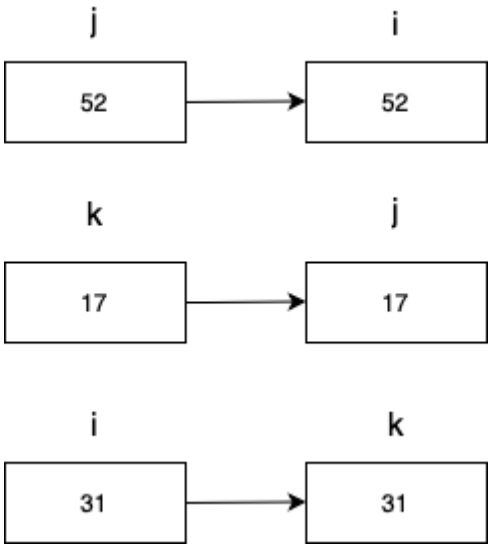
1. Dans un premier temps on initialise les pointeurs et les variables.



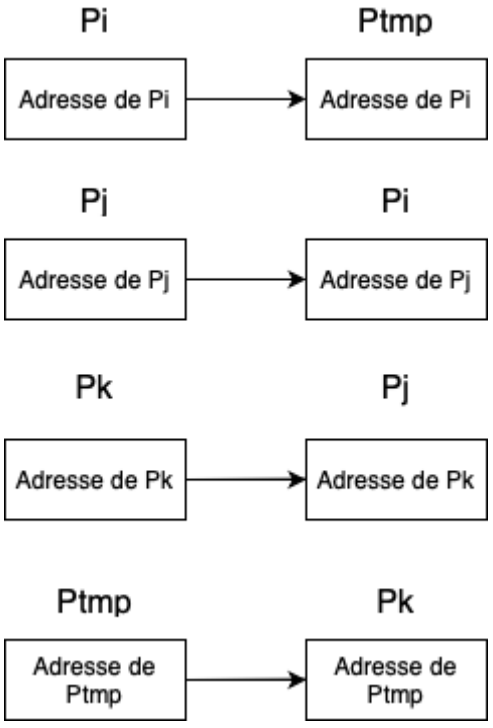
2. On affecte les valeurs 31, 52, 17 respectivement aux variables i, j, k et les pointeurs Pi, Pj, Pk vers leurs variables respectives.



3. L'opérateur * devant un pointeur, permet de changer la valeur de la variable qui se trouve à l'adresse contenue dans ce pointeur.
Dans la deuxième affectation, on change la valeurs contenue dans les variables i, j, k mais en passant par leurs pointeurs avec l'opérateur *.



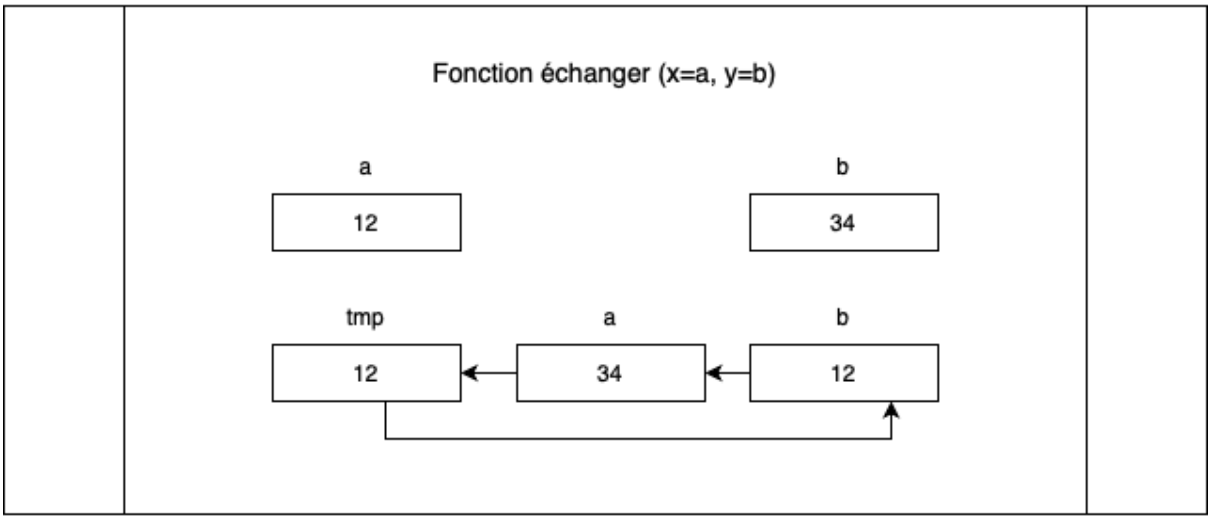
4. Dans la dernière affectation, comme on ne mets plus * devant les pointeurs, ce sont donc les adresses des pointeurs qui sont changées. Le résultat est que Pj pointe sur k, Pk pointe sur tmp et Pi pointe sur j.



III) Écrire une procédure qui modifie la valeur de ses arguments

Les valeurs de x et y sont passées par valeur, tel quel :

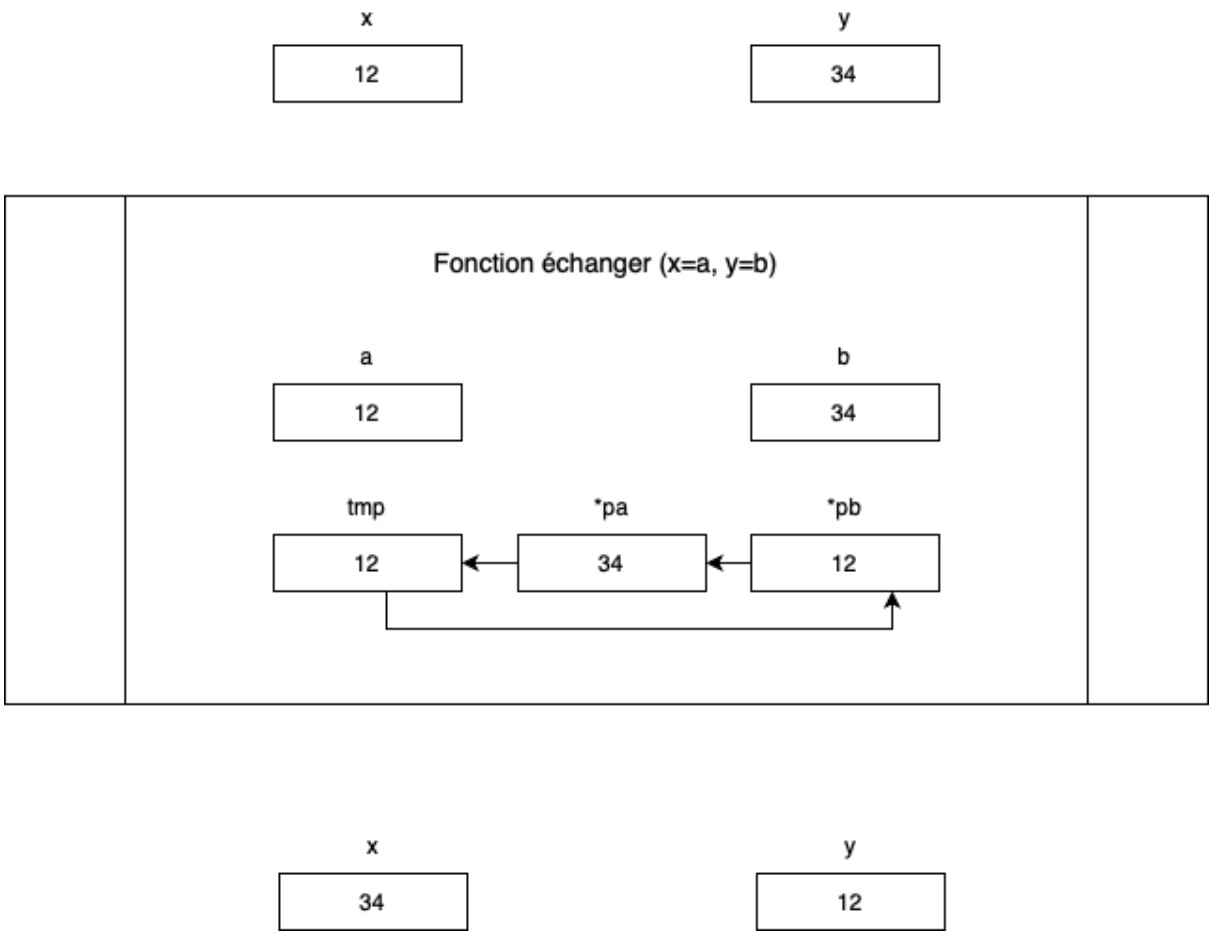
- a et b sont des copies de x et y, pas leurs vraies adresses.
- Les modifications faites sur a et b dans la fonction echanger() ne changent pas x et y dans le main().



Après avoir corrigé le code :

- &x et &y passent les adresses des variables x et y à echanger().
- pa et pb sont des pointeurs qui stockent ces adresses.
- *pa et *pb permettent d'accéder aux valeurs pointées et de les modifier directement.

Après l'échange, les valeurs de x et y sont bien modifiées. Pour aller plus loin :



- Avant l'appel à echanger(&x, &y);
  - x = 12, y = 34.
- Dans la fonction echanger(int *pa, int *pb)
  - tmp = *pa; (Stocke 12 dans tmp).
  - *pa = *pb; (x prend la valeur de y, donc x = 34).
  - *pb = tmp; (y prend la valeur initiale de x, donc y = 12).
- Après l'appel à la fonction, x et y sont bien échangés.

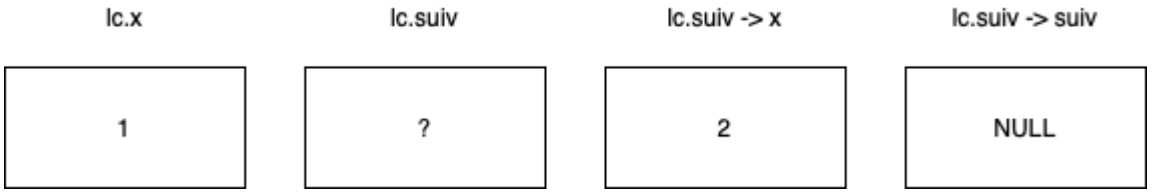
III) Introduire des doubles pointeurs

La fonction reçoit une copie de p, donc quand p change de valeur (p = &y;), c'est seulement la copie qui change, pas l'originale. C'est le même mécanisme que ce que nous avons vu à l'exercice précédent.

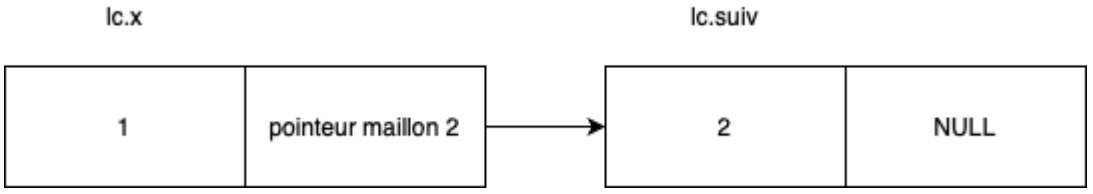
IV) met en application la notion de pointeur – Listes chaînées

1. Le programme initialise une structure maillon et crée dynamiquement un second maillon en mémoire avec malloc(). Il stocke les valeurs 1 et 2 dans les champs x respectifs. Chaque maillon est une structure d'une valeur et d'un pointeur qui pointe sur le prochain maillon.

Un schéma du programme :



Un schéma de la mémoire :



2.

2. Écrire le programme suivant et expliquer à l'aide d'un schéma son fonctionnement et sa représentation mémoire.

Voici une explication du programme, qui a été modifié en ajoutant une structure Maillon permettant la bonne compilation et exécution de celui-ci.
Dans ce programme, chaque maillon contient un entier 'x' et un pointeur 'suiv' vers le maillon suivant. Le dernier maillon de la liste a 'suiv' = NULL`, indiquant la fin de la liste.
Le programme se déroule de la manière suivante:

1. Création du premier maillon (tete) avec allocation dynamique.
2. Boucle de création des maillons suivants en allouant dynamiquement de la mémoire et en reliant les maillons entre eux.
3. Remplissage des champs x des maillons avec des valeurs de 0 à N-1.
4. Affichage des valeurs des maillons et des adresses mémoire des pointeurs.
5. Libération de la mémoire allouée pour éviter les fuites.

Voici un schéma représentant les valeurs mémoires