Travaux Pratiques n°1

Cours d'Informatique de Deuxième Année

—Licence MI L2.2—

Allocation dynamique, listes chaînées

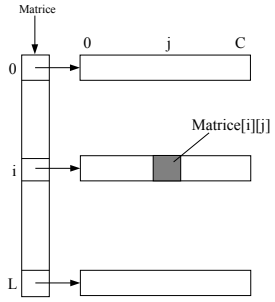
Cette feuille d'exercice a pour but de rappeler rapidement les principes suivants : allocation et désallocation de mémoire, manipulation des pointeurs (arithmétique), structures récursives, passage de paramètre par adresse. Pour illustrer ces notions, nous nous appuierons, d'une part, sur l'exemple des tableaux à 2 dimensions et, d'autre part, sur l'exemple des listes simplement chaînées.

▸ Exercice 1. *Matrice et typage de la mémoire*

On désire représenter par un pointeur une matrice d'éléments de type `double` *(tableau à deux dimensions). Le faire par une allocation statique est plutôt simple :*

```
double tab[MAXLIG][MAXCOL];
```

Pour allouer dynamiquement un tel tableau, il est possible de s'aider de la figure ci-dessous qui présente la manière dont les informations devront être stockées.



Écrire les instructions nécessaires à l'allocation dynamique d'un tableau de `MAXLIG` *lignes et* `MAXCOL` *colonnes d'éléments de type* `double`*, représenté par un pointeur.*

Écrire une fonction `double element()` *qui accepte en argument un pointeur sur une matrice m et deux indices i et j, et qui retourne l'élément correspondant à* `m[i][j]` *sans utiliser cette expression ni aucun accès utilisant les crochets (*`[]`*), mais en utilisant l'arithmétique sur les pointeurs.*

▸ **Exercice 2.** ***Type structure***

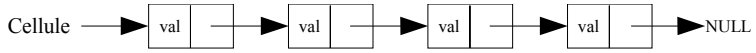
Donner la définition d'un type `Matrice` *permettant de représenter des matrices à deux dimensions d'éléments de type* `double`*, dont la taille de chaque dimension n'est pas connue statiquement. Les informations nécessaires sont donc un pointeur et la taille de chaque dimension.*

Écrire les fonctions :

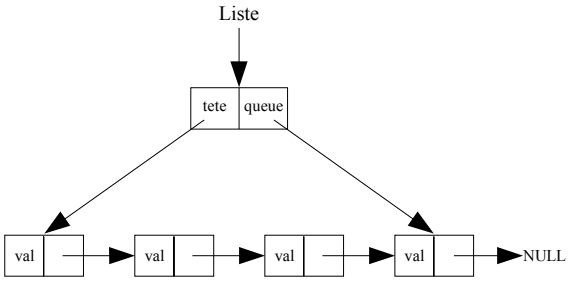
- `Matrice *allalloueMatrice(int maxLig, int maxCol);`
- `double elementMatrice(Matrice * m, int i, int j);`
- `void afficheMatrice(Matrice * m);`
- `void libereMatrice(Matrice * m);`

▸ **Exercice 3.** ***Type structure récursive***

Définir le type `Cellule` *nécessaire à la gestion d'une liste chaînée d'entiers. On rappelle que leur implantation peut être illustrée par le schéma ci-dessous.*



On désir encapsuler cette liste chaînée par une structure un peu plus générale nous permettant de ne connaître que le début et la fin de la liste comme sur le schéma ci-dessous :



Définir le type `Liste` *correspondant.*

Implémenter les fonction suivantes :

- `allalloueCellule()` *qui alloue une cellule contenant la valeur passée en paramètre ;*
- `libereCellule()` *réalisant la désallocation d'une cellule passée en paramètre ;*
- `estVide()` *acceptant une liste et retourant 1 si la liste est vide et 0 sinon ;*
- `insereEnTete()` *acceptant une liste et une valeur, et insérant une nouvelle cellule ayant cette valeur en tête de la liste ;*
- `insereEnQueue()` *acceptant une liste et une valeur, et insérant une nouvelle cellule ayant cette valeur à la fin de la liste ;*
- `supprimeTete()` *acceptant une liste, supprimant la cellule en tête de liste et retournant la valeur de cette liste.*

▸ **Exercice 4.** ***A la maison...***

1 Cube

Reprendre l'exercice 2 et adapter la structure de la matrice en ajoutant une dimension de sorte à avoir un "Cube" (un tableau à trois dimensions) et écrire les fonctions :

- `Cube * alloueCube(int maxLong, int maxHaut, maxProf);`
- `void libereCube(Cube * c);`
- `double elementCube(Cube * c, int i, int j, int k);`
- `void afficheCube(Cube * c);`

2 Liste simplement chaînée

Implémenter les fonctions suivantes pour gérer la liste définie précédement :

Pour chacune des fonctions, préciser les instructions d'appel et justifier l'usage des pointeurs.

1. `allouCetteCellule()`, *effectue l'allocation d'une cellule qu'elle a reçu en argument ;*
2. `afficheListeIt()` *prenant en argument une liste et affiche cette liste itérativement ;*
3. `afficheListeRec()` *prenant en argument une liste appelant une autre fonction qui parcours et affiche la liste des cellule récursivement ;*
4. *Écrire une fonction* `libereListe()` *réalisant la désallocation de toutes les cellules d'une liste ;*
5. `supprimeQueue()` *acceptant une liste, supprimant la cellule en fin de liste et retournant la valeur de cette liste ;*
6. *Écrire une fonction* `insereEnPlace()` *acceptant une liste et une valeur, mais qui insère la nouvelle cellule de sorte que la liste reste triée par ordre croissant des valeurs entières ;*
7. *Écrire une fonction* `recherche()` *qui, étant donnés une liste et une valeur entière, retourne un pointeur sur la cellule de la liste contenant cette valeur, ou* `NULL` *si la valeur n'est pas dans la liste ;*
8. *Écrire une fonction* `supprime()` *qui, étant donnés une liste et une valeur entière, supprime la première cellule de la liste contenant cette valeur. Si une telle cellule est supprimée, la fonction retourne 1, sinon elle retourne 0.*