II - Éléments avancés

A_ Tableaux multidimensionnels

Rapel Un *tableau* est une structure de données rassemblant des éléments de même type.

1. Allocation statique

voir cours déjà donné

2. Allocation dynamique - tableau 1D

voir cours déjà donné

Syntaxe Allouer à l'exécution un tableau contenant `n` éléments de type `int64_t` :

```
int64_t* p = (int64_t*)malloc(sizeof(uint64_t)*n);
// traitements
free(p);
```

Vocabulaire Effectuer un *cast*, c'est effectuer une opération de conversion d'un type vers un autre.
Dans l'exemple vu en cours, `malloc` renvoie l'adresse de la zone mémoire allouée sous la forme d'un `void *` que l'on *cast* en `int*`.
Pour allouer à l'exécution un tableau multidimensionnel, il faut déterminer sa représentation et sa taille en mémoire.

Règle toute zone mémoire allouée doit être libérée !

3. Allocation d'un bloc contigü

Comme on l'a vu en cours, la zone mémoire allouée par `malloc` est une zone linéaire dans la mémoire.
Pour représenter les données d'un tableau $M \times N$, le bloc de données prendra une taille en mémoire de : $M \times N \times s$ où s est la taille du type des éléments.
Ce sera au programmeur de gérer la manière dont il accède aux données.
Reprenons l'exemple d'une matrice :

```
int m = 25;
int n = 10;
double* matrice = (double*)malloc(sizeof(double)*m*n);
if (matrice == NULL)
{
  exit(EXIT_FAILURE);
}
// Accès à un élément d'indice i, j
double valeur = matrice[i*n+j];

// traitements
free(matrice);
```

Plus généralement, pour $i \in [0;m[$ et $j \in [0;n[$ toute fonction bijective qui prend un couple (i,j) et renvoie un entier a tel que $a \in [0;m \times n[$ convient comme fonction d'accès.

Exercice Si on utilse la fonction d'accès $f(i,j) -> i + n \times j$, déterminer les coordonnées (i,j) lorsqu'on donne $a \in [0;m \times n[$.

4. Allocation de tableaux multidimensionnels avec indirection

Une deuxième possibilité est de déclarer un tableau d'indirection contenant des pointeurs qui contiennent chacun l'adresse de zones en mémoire différentes.

Reprenons le cas d'un tableau 2D contenant des `double` que l'on souhaite stocker en mémoire et accéder à l'aide d'un tableau de pointeurs.

Nous devons allouer :
- une zone en mémoire pour un premier tableau contenant les adresses des tableaux de second niveau
- pour chaque tableau du second niveau
  - une zone en mémoire pour contenir les données correspondant au tableau

```c
int m = 25;
int n = 10;
double **matrice = (double**)malloc(sizeof(double*)*m);
if (matrice == NULL)
{
  exit(EXIT_FAILURE);
}
for(int i = 0; i < m; i++)
{
  matrice[i] = (double*)malloc(sizeof(double)*n);
  if (matrice[i] == NULL)
  {
    exit(EXIT_FAILURE);
  }
}

// traitements

// libération de la mémoire
for(int i = 0; i < m; i++)
{
  free(matrice[i]);
}
free(matrice);
```