Langace C - Travaux Dirigés n° 7

— Licence Informatique —

Deux types différents `Array2d` et `Array2D` ont été introduits à la séance précédente pour programmer des tableaux dynamiques à deux dimensions en C. L'inconvénient de cette approche est qu'un programme qui souhaite utiliser ces tableaux doit choisir entre l'un des deux types. Si par exemple, ce programme a une fonction `f` qui prend un tableau `a` à deux dimensions en paramètre, le type de ce paramètre `a` sera `Array2d` ou `Array2D`. Si le programme doit utiliser simultanément des tableaux de type `Array2d` et des tableaux de type `Array2D`, il devra comporter deux fonctions `f`, une pour chacun des deux types. Le but de ce Td est de voir une méthode permettant de contourner cette difficulté.

Les trois opérations essentielles à effectuer sur les tableaux à deux dimensions sont

1. la création,
2. l'accès à un élément du tableau,
3. la destruction.

Toutes les autres opérations peuvent être écrites en utilisant ces trois opérations de bases. Dans un premier temps, on met de côté la destruction et on ne considère que la création et l'accès à un élément.

L'idée de base est de créer un type qui va simuler les deux types `Array2d` et `Array2D`. On définit donc le type `Array2` de la manière suivante.

```
struct AA {
  void* vec;                    // Pointeur vers le tableau ou le vecteur
  int lin;                      // Nombre de lignes du tableau
  int col;                      // Nombre de colonnes du tableau
  int& (*access)(AA&, int, int);// Accesseur
};
typedef AA Array2;
```

Un objet de type `Array2` peut en effet contenir un objet de type `Array2d` ou un objet de type `Array2D`. Les champs `lin` et `col` contiennent comme d'habittude le nombre de lignes et de colonnes du tableau. Le champ `vec` contient le pointeur qui est dans le champ `tab` du type `Array2d` ou dans le champ `vec` du type `Array2D`. Ceci est possible puisqu'une variable de type `void*` peut contenir une valeur de type `int*` ou une valeur de type `int**`.

L'accès à un élément du tableau ne se fait bien sûr pas de la même façon suivant que l'objet de type `Array2` simule un objet qui serait de type `Array2d` ou un objet qui serait de type `Array2D`. Ceci est pris en compte par le champ `access`. C'est un pointeur vers une fonction chargée de réaliser l'accès à un élément. Cette fonction prend en paramètre le tableau et les indices de ligne et de colonne et rend une référence sur l'élément.

▶ Exercice 1 (Méthodes d'allocation)

Écrire des fonctions `arrayAccess2d` et `arrayAccess2D` qui réalisent l'accès à un élément pour un objet de `Array2` qui simule respectivement un objet de type `Array2d` ou un objet de type `Array2D`.

▶ **Exercice 2 (Méthodes d'accès à un élément)**
Écrire des fonctions `arrayAlloc2d` et `arrayAlloc2D` qui réalisent l'allocation d'un objet de `Array2` qui simule respectivement un objet de type `Array2d` ou un objet de type `Array2D`. Le champ `access` doit être correctement positionné par ces fonctions.

▶ **Exercice 3 (Accès à un élément)**
Écrire une fonction `arrayAccess` qui réalise l'accès à un élément du tableau en utilisant le champ `access`.

▶ **Exercice 4 (Destruction)**
Le problème de la destruction des tableau a été négligé. Résoudre ce problème.

▶ **Exercice 5 (Matrices)**
Écrire des fonctions de manipulation de matrices. Le type des matrices est un des deux types de tableaux précédents. On le définit par exemple par

```
typedef Array2 Matrix;
```

Les opérations à définir sont

1. la saisie et l'affichage d'une matrice.
2. la somme et la différence de deux matrices.
3. le produit de deux matrices.
4. la puisssance n-ième d'une matrice en utlisant l'exponentiation logarithmique.

Utiliser ces fonction pour calculer le coefficient de $M^n_{0,0}$ où M est la matrice 2×2 donnée par

$$M = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$$

Que vaut ce coefficient ?