Tableaux à deux indices

Lycée Thiers

Crédits

- Cette page de CommentCaMarche.net

Crédits

- Cette page de CommentCaMarche.net
- ≪ Langage C ≫ -Claude Delannoy- Ed. Eyrolles

Déclaration de tableaux à deux indices

- Les tableaux multidimensionnels sont des tableaux qui contiennent des tableaux.

Déclaration de tableaux à deux indices

- Les tableaux multidimensionnels sont des tableaux qui contiennent des tableaux.
- Un déclarateur de tableau est de la forme `decl1 [nb1]`. Si `decl1` est lui-même un déclarateur de tableau, il peut être de la forme `decl2 [nb2]`.
  Ainsi, pour déclarer un tableau de tableaux, on utilise la syntaxe `decl2 [nb2] [nb1]`.

Déclaration de tableaux à deux indices

- Les tableaux multidimensionnels sont des tableaux qui contiennent des tableaux.
- Un déclarateur de tableau est de la forme `decl1 [nb1]`. Si `decl1` est lui-même un déclarateur de tableau, il peut être de la forme `decl2 [nb2]`.
  Ainsi, pour déclarer un tableau de tableaux, on utilise la syntaxe `decl2 [nb2] [nb1]`.
- Exemple : `int tab [5] [3];`. Dans ce cas :

Déclaration de tableaux à deux indices

- Les tableaux multidimensionnels sont des tableaux qui contiennent des tableaux.
- Un déclarateur de tableau est de la forme `decl1 [nb1]`. Si `decl1` est lui-même un déclarateur de tableau, il peut être de la forme `decl2 [nb2]`.
  Ainsi, pour déclarer un tableau de tableaux, on utilise la syntaxe `decl2 [nb2] [nb1]`.
- Exemple : `int tab [5] [3];`. Dans ce cas :
  - `tab [4][2]` est un `int`

Déclaration de tableaux à deux indices

- Les tableaux multidimensionnels sont des tableaux qui contiennent des tableaux.
- Un déclarateur de tableau est de la forme `decl1 [nb1]`. Si `decl1` est lui-même un déclarateur de tableau, il peut être de la forme `decl2 [nb2]`.
  Ainsi, pour déclarer un tableau de tableaux, on utilise la syntaxe `decl2 [nb2] [nb1]`.
- Exemple : `int tab [5] [3];`. Dans ce cas :
  - `tab [4][2]` est un `int`
  - `tab [4]` est un tableau de 3 `int`

Déclaration de tableaux à deux indices

- Les tableaux multidimensionnels sont des tableaux qui contiennent des tableaux.
- Un déclarateur de tableau est de la forme `decl1 [nb1]`. Si `decl1` est lui-même un déclarateur de tableau, il peut être de la forme `decl2 [nb2]`.
  Ainsi, pour déclarer un tableau de tableaux, on utilise la syntaxe `decl2 [nb2] [nb1]`.
- Exemple : `int tab [5] [3];`. Dans ce cas :
  - `tab [4][2]` est un `int`
  - `tab [4]` est un tableau de 3 `int`
  - `tab` est un tableau dont chaque élément est lui-même un tableau de 3 `int`.

Organisation en mémoire

Considérons `int tab [5][3]`.

- Avec cette déclaration `tab` n'est pas un tableau de pointeurs : les éléments du tableau sont ici 15 entiers rangés consécutivement en mémoire.

Organisation en mémoire

Considérons `int tab [5][3]` .

- Avec cette déclaration `tab` n'est pas un tableau de pointeurs : les éléments du tableau sont ici 15 entiers rangés consécutivement en mémoire.
- Accès : Pour accéder à l'élément d'indice (i, j), le compilateur calcule un décalage de $4 \times (\underbrace{3}_{\text{nb. de col.}} \times i + j)$ octets par rapport à l'adresse du premier élément du tableau.
  On observe que, pour localiser un élément du tableau, la grandeur vraiement importante est le nombre de « colonnes » pas le nombre de lignes (ce mot *colonne* ne correspond d'ailleurs à aucune réalité informatique, mais il est bien connu des mathématiciens)

Débordement d'indices

Considérons `int tab [5][3]`

- Les éléments sont rangés dans des emplacements contiguës de la mémoire :

```
tab[0][0], tab[0][1], tab[0][2], tab[1][0], tab[1][1], tab[1][2], tab[2][1]...
```

Débordement d'indices

Considérons `int tab [5][3]`

- Les éléments sont rangés dans des emplacements contiguës de la mémoire :

  `tab[0][0], tab[0][1], tab[0][2], tab[1][0], tab[1][1], tab[1][2], tab[2][1]...`

- Ainsi `tab[0][5]` désigne `tab[1][2]`

Débordement d'indices

Considérons `int tab [5][3]`

- Les éléments sont rangés dans des emplacements contiguës de la mémoire :

  `tab[0][0], tab[0][1], tab[0][2], tab[1][0], tab[1][1], tab[1][2], tab[2][1]...`

- Ainsi `tab[0][5]` désigne `tab[1][2]`
- Et `tab[5][0]` désigne un élément juste au delà des limites du tableau.

Passage en paramètres

Considérons `int tab [5][3]`

- Si on doit passer un tableau multidimensionnel en paramètre d'une fonction, il est important de communiquer au moins la deuxième dimension.
  On écrit donc `void f(int tab[5][3])` ou même `void f(int tab[][3])`, dans la mesure où `5` n'a qu'une valeur informative tandis que `3` est indispensable à la localisation des éléments.

Passage en paramètres

Considérons `int tab [5][3]`

- Si on doit passer un tableau multidimensionnel en paramètre d'une fonction, il est important de communiquer au moins la deuxième dimension.
  On écrit donc `void f(int tab[5][3])` ou même `void f(int tab[][3])`, dans la mesure où `5` n'a qu'une valeur informative tandis que `3` est indispensable à la localisation des éléments.
- Dans le cas d'un tableau multidimensionnel dont la taille n'est pas connue à la compilation, on utilise plutôt un tableau de pointeurs et on écrit `void f(int n, int m, int **tab);`.
  On fait l'hypothèse ici que `tab` est un tableau de n pointeurs, chacun représentant un tableau de m entiers.
  Avec `**tab` en paramètre, les « lignes » de la matrice ne sont plus contiguës en mémoire.