Tableaux « à la C »

V. Lepetit, J. Sam, et J.-C. Chappelier

Comme mentionné dans le cours sur les tableaux, en plus des types `vector` et `array`, on peut utiliser en C++ des tableaux « à la C », c'est-à-dire les tableaux comme définis par le langage C.

Ces tableaux sont des tableaux de taille fixe, et ont de nombreuses limitations. Contrairement aux types `vector` et `array` :

- ils n'ont pas connaissance de leur propre taille, c'est-à-dire qu'on ne dispose pas de fonction similaire à la fonction `size` ;
- ils ne peuvent être manipulés globalement, c'est-à-dire qu'on ne peut par exemple pas faire d'affectations telles que : `tab1 = tab2;` ;
- ils ne peuvent pas être retournés par une fonction ;
- ils ont une syntaxe d'initialisation différente des autres types, c'est-à-dire qu'on ne peut pas utiliser la syntaxe avec des parenthèses comme `({1, 2, 3})`.

De plus, quand ils sont utilisés en paramètre d'une fonction, ils sont forcément passés par référence.

Cependant, ils sont encore souvent utilisés en pratique.

Déclaration et initialisation

La déclaration d'un tableau « à la C » suit la syntaxe suivante :

type_des_éléments nom_du_tableau`[`*nombre_d_éléments*`];`

Par exemple :

```
int T[5];
```

déclare un tableau `T` de 5 `int`.

On peut initialiser les éléments d'un tableau « à la C » en faisant par exemple :

```
int T[5] = {1, 2, 3, 4, 5};
```

La taille des tableaux « à la C » doit être connue à la compilation, c'est-à-dire qu'on ne peut pas écrire par exemple :

```
cin >> N;
int T[N];  // NON !
```

Accéder aux éléments du tableau

Les éléments d'un tableau « à la C » se manipulent de la même façon que les éléments des types `vector` et `array`, par exemple :

```
T[3] = 10;
T[2] = 2 * T[3] + T[1];
```

Comme pour `vector` et `array`, l'indice du premier élément est 0.

En C++11, on peut utiliser les itérations sur ensemble de valeurs comme pour les types `vector` et `array`, par exemple :

```
int T[3] = {1, 2, 3};

for(int val : T) {
  cout << val << ", ";
}
```

Taille du tableau

Il n'existe pas d'équivalent de la fonction `size` pour les tableaux « à la C », c'est-à-dire qu'on ne peut pas écrire quelque chose comme `T.size()` pour obtenir la taille du tableau.

En pratique, on utilise souvent une constante pour définir la taille du tableau, par exemple :

```
const int NB(3);

int T[NB] = {1, 2, 3};

cout << "Dernier element de T : " << T[NB - 1] << endl;
```

Attention

Attention à ne pas confondre :
- la déclaration d'un tableau « à la C », et
- l'accès à un élément du tableau.

Les deux s'écrivent avec le nom du tableau suivi d'une valeur entre crochets, mais ils font des choses très différentes.

```
int T[5];
```

est une déclaration de tableau. `5` est la taille du tableau : on déclare un tableau de 5 éléments.

```
T[2] = 0;
```

accède à un élément. `2` est l'indice de l'élément : on affecte 0 à l'élément d'indice 2.

Tableaux multi-dimensionnels « à la C »

Les tableaux « à la C » peuvent avoir plusieurs dimensions. Par exemple :

```
int M[3][4][4];
int N[3][2] = { {1, 2}, {3, 4}, {5, 6} };

M[1][2][0] = N[1][0];
```

Tableaux « à la C » en paramètre

Un tableau « à la C » en paramètre est forcément passé par référence, SANS utiliser le symbole `&`. Par exemple, la fonction :

```
void f(int T[5])
{
  for(int i(0); i < 5; ++i) {
    T[i] = T[i] + 1;
  }
}
```

ajoute 1 aux éléments du tableau passé en argument. Le code :

```
int N[5] = { 4, 3, 1, 0, 2 };

f(N);
for(int i(0); i < 5; ++i) {
  cout << N[i] << ", ";
}
```

affiche :

```
5, 4, 2, 1, 3,
```

En fait, le `5` dans l'en-tête de la fonction `f`

```
void f(int T[5])
```

n'est pas utilisé par le compilateur. On peut également écrire :

```
void f(int T[])
```

et aussi :

```
void f(int * T)
```

avec exactement le même résultat.

La raison sort du cadre de ce cours (mais notez qu'elle fait appel aux pointeurs « à la C » qui sont mentionnés lors du cours de la semaine 6).

La fonction `f` ne connait donc pas la taille du tableau passé en argument.

En pratique, la taille du tableau est généralement passée en argument aussi, par exemple :

```
void f(int T[], int taille)
{
  for(int i(0); i < taille; ++i) {
    T[i] = T[i] + 1;
  }
}

 ...

  int N[5] = { 4, 3, 1, 0, 2 };
  f(N, 5);

  int N2[3] = { 3, 0, 2 };
  f(N2, 3);
```