Tableaux

1 Introduction

Les tableaux sont des variables analogues aux variables indicées des mathématiques. Par exemple ce qui s'écrit x_i en mathématiques s'écrira `x[i]` en C, a_{ijk} : `a[i][j][k]`. L'avantage est le même qu'en mathématiques, on a une écriture plus homogène et surtout beaucoup plus systématique.
La seule difficulté pour utiliser des tableaux est qu'il faut les déclarer comme tels en précisant la limite supérieure (entière) que chaque indice ne dépassera pas durant l'exécution. Par exemple :

```
double x[20];
```

indique que le tableau `x` est de type double et a 20 éléments indicés de 0 à 19 (en C l'indice commence toujours à 0). Cette déclaration indique qu'une certaine variable est un tableau et non une simple variable et réserve le nombre de places mémoire nécessaires.

Exemple

calcul du produit scalaire de deux vecteurs sans utiliser de tableau :

```
#include<iostream>
using namespace std;
int main()
{
double ux,uy,uz,vx,vy,vz,s;
...
s=ux*vx+uy*vy+uz*vz;
...
return 0;
```

et en utilisant un tableau :

```
#include<iostream>
using namespace std;
#define D 3
int main()
{
double u[D],v[D];
int i;
...
for(s=0.,i=0; i<D; i++) s=s+u[i]*v[i];
...
return 0;
}
```

La seconde écriture paraît plus compliquée mais elle est plus générale : si on veut calculer dans un espace à plus de trois dimensions, il suffit, dans le second cas, de changer la valeur de `D` dans le `#define`, alors que cela conduit, dans le premier cas, à créer de plus en plus de noms variables et à les écrire explicitement dans les opérations ce qui peut devenir très lourd voire irréalisable.

2 Déclaration des tableaux

La déclaration de tableau se place comme toute déclaration en début de programme et elle a la forme générale, pour un tableau de réels par exemple :

```
double t[N1][N2][N3][...];
```

`N1, N2, N3, ...` sont des constantes entières explicites ou définies par un `#define`, comme dans `#define N1 20` par exemple, ou des expressions formées de telles constantes. Par contre, il est impossible de définir ces constantes par un `const` comme dans `const int n1=20` [1].

1. C'est possible en C++, mais cela n'a pas d'avantage flagrant.

Ces constantes signifient que :
le premier indice pourra varier entre les valeurs `0` et `N1-1`
le second indice pourra varier entre les valeurs `0` et `N2-1`
...

Remarque

Les tableaux peuvent être des trois types vus jusqu'à présent c'est à dire : entier, réel et caractère.

Remarque

Les valeurs des indices peuvent être le résultat d'une expression arithmétique quelconque, par exemple :

```
t[2*i][j*j+k][j/k+l%m]
```

2.1 Dépassement de la taille déclarée

Considérons le programme suivant :

```
#include<iostream>
using namespace std;
int main()
{
double x[4],y[4],r=10.; int i;
for(i=0; i<4; i++) {x[i]=i; y[i]=r+i;}
for(i=0; i<4; i++) {cout << x[i] << " " << y[i] << endl;}
return 0;
}
```

dont le résultat est :

```
0 10
1 11
2 12
3 13
```

Si on déclare `x` et `y` comme de simples variables et non des tableaux :

```
double x,y,r=10.; int i;
```

au lieu de :

```
double x[4],y[4],r=10.; int i;
```

on a le diagnostic suivant à la compilation : *erreur : invalid types 'double[int]' for array subscript*
Mais si on déclare une mauvaise dimension, par exemple :

```
double x[2],y[2],r=10.; int i;
```

au lieu de :

```
double x[4],y[4],r=10.; int i;
```

il n'y a pas de diagnostic à la compilation mais un diagnostic à l'exécution : *Erreur de segmentation.*

2.2 Initialisation des tableaux

2.2.1 Tableaux à un indice

Les tableaux, comme les variables simples, ne sont pas initialisés[2]. Leur contenu initial peut donc être n'importe quoi.
Supposons que l'on veuille initialiser le tableau `t` à trois éléments avec les valeurs : `t[0]=9, t[1]=19, t[2]=29`, par exemple. L'initialisation se fait en même temps que la déclaration par :

```
int t[3]={9,19,29}
```

2. On verra plus bas que c'est le cas de toutes les variables dites ≪ automatiques ≫ .

Les valeurs placées entre accolades ne peuvent être que des constantes explicites ou des constantes définies par `#define`. Ce peut être des expressions constituées de telles constantes. On obtient par exemple le même résultat que précédemment avec :

```
#define N 10
int t[3]={N-1,2*N-1,3*N-1}
```

On peut toujours, sinon, initialiser les tableaux par des instructions exécutables.

2.2.2 Tableaux à plusieurs indices

Pour un tableau à plusieurs indices il faut savoir que le compilateur range les éléments du tableau en faisant varier le dernier indice en premier, l'avant dernier en second, etc. jusqu'au premier indice en dernier. Par exemple pour le tableau déclaré par :

```
int t[2][3]
```

les éléments sont rangés dans l'ordre :

```
t[0][0] t[0][1] t[0][2] t[1][0] t[1][1] t[1][2]
```

Si l'on veut par exemple initialiser aux valeurs :

```
t[0][0]=0 t[0][1]=1 t[0][2]=2 t[1][0]=3 t[1][1]=4 t[1][2]=5
```

on écrit soit :

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

soit :

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

(on va à la ligne uniquement pour la lisibilité).
Comme dans le cas à un indice les valeurs placées entre accolades ne peuvent être que des constantes explicites ou des constantes définies par `#define` ou des expressions constituées de telles constantes.

3 Affichage d'un tableau

3.1 Tableau à un indice (vecteur)

```
/* Affichage d'un tableau a un indice */
#include<iostream>
using namespace std;
int main()
{
double x[4]={0.,1.,2.,3.};
int i;
cout << endl << endl; // on separe de ce qui precede par deux lignes blanches
for(i=0; i<4; i++) cout << x[i] << "  ";
cout << endl << endl << endl; /* on separe de ce qui suit par deux lignes blanches */
return 0;
}
```

3.2 Tableau à deux indices (matrice)

```
/* Affichage d'un tableau a deux indices */
#include<iostream>
using namespace std;
int main()
{
double xx[2][3]={{4,5,6},
                 {7,8,9}};
int i,j;
```

```
cout << endl << endl;
for(i=0; i<2; i++)
   {
   for(j=0; j<3; j++) cout << xx[i][j] << "  ";
   cout << endl;
   }
cout << endl << endl;
return 0;
}
```

3.3 Tableau à trois indices (tenseur d'ordre trois)

```
/* Affichage d'un tableau a trois dimensions */
#include<iostream>
using namespace std;
int main()
{
double xxx[2][2][3]={{{10,11,12},
                      {13,14,15}},
                     {{16,17,18},
                      {19,20,21}}};
int i,j,k;
cout << endl << endl;
for(i=0; i<2; i++)
   {
   for(j=0; j<2; j++)
      {
      for(k=0; k<3; k++) cout << xxx[i][j][k] << "  ";
      cout << endl;
      }
   cout << endl;
   }
cout << endl;
return 0;
}
```

4 Remarques et exemples

Il ne faut utiliser un tableau que lorsque c'est réellement nécessaire, c'est à dire quand il faut conserver toutes les valeurs calculées pour les réutiliser en un autre point du programme. Sinon on mobilise des mémoires pour rien et, de plus, on subit les contraintes liées à la déclaration.

Exemple

on veut calculer les n premiers termes de la suite : $u_n = au_{n-1} + bu_{n-2}$, connaissant u_0 et u_1.

1) Avec un tableau :

```
/* Calcul de la suite u(n)=a*u(n-1)+b*u(n-2) en utilisant inutilement un tableau */
#include<iostream>
using namespace std;
#define NN 50
int main()
{
  double a=5.6, b=-3.4, u[NN]; int i;
  u[0]=-4.2; u[1]=7.1;
  for(i=2;i<NN;i++)
    {
      u[i]=a*u[i-1]+b*u[i-2];
```

```
    }
  for(i=0;i<NN;i++)
    {
      cout << "u(" << i << ")=" << u[i] << endl;
    }
  return 0;
}
```

2) Sans tableau :

```
/* Calcul de la suite u(n)=a*u(n-1)+b*u(n-2) sans utiliser de tableau */
#include<iostream>
using namespace std;
#define NN 50
int main()
{
  double a=5.6, b=-3.4, u, u0=-4.2, u1=7.1; int i;
  cout << "u(0)=" << u0 << endl;
  cout << "u(1)=" << u1 << endl;
  for(i=2;i<NN;i++)
    {
      u=a*u1+b*u0;
      cout << "u(" << i << ")=" << u << endl;
      u0=u1; u1=u;
    }
  return 0;
}
```

La seconde méthode est plus simple.
Mais si on a besoin des u_n pour une tâche qui ne peut être effectuée au vol (par exemple les faire afficher par valeur croissante) il faut utiliser la première méthode.
On dit que dans le premier cas on a un accès direct aux u_n alors que dans le second on a seulement un accès séquentiel.

Autre exemple

On veut calculer la moyenne et l'écart-type d'un ensemble de valeurs entrées au clavier par l'utilisateur :

```
#include<iostream>
using namespace std;
#include <math.h>
int main()
{
  int i,n; double s,s2,x,moy;
  cout << "Nombre de valeurs ? "; cin >> n;
  for(s=0,s2=0,i=1;i<=n;i++)
    {
      cin >> x;
      s+=x; s2+=x*x;
    }
  moy=s/n;
  cout << "moyenne=" << moy << "  écart-type=" << sqrt(s2/n-moy*moy) << endl;
  return 0;
}
```

On voit qu'il est inutile de stocker les valeurs dans un tableau.