DM1 MP2I

Tableaux de tableaux

Cette présentation est inspirée d'une page de Waheed ASLAM.

Le langage C permet de définir des tableaux à plusieurs dimensions. L'opérateur `[ ]` peut être juxtaposé à un autre opérateur pour définir des tableaux de tableaux.

— `char ecran[24][80]` : tableau composé de 24 lignes et de 80 colonnes. Ce tableau est complété par des caractères, d'où la syntaxe char.

— `int matrice[100][100]` : tableau formé de 100 tableaux constitués chacun de 100 entiers. Elle réserve une zone de 100*100=10000 entiers.

— `int matrices[10][10][10]` : 10 tableaux de tableaux de 10*10 d'entiers.

Un tableau à plusieurs dimensions peut être initialisé, soit lors de sa déclaration, soit au cours de la programmation. Les deux types d'initialisations suivantes sont équivalentes :

```c
//déclaration indiquant la structure par ligne
int tab[2][5]={{0,1,2,3,4},{4,3,2,1,0}};

//déclaration en une ligne
int t[2][5]={0,1,2,3,4,4,3,2,1,0}
```

La première méthode fait apparaître les lignes de la « matrice » mais pas la seconde. Cependant, en mémoire tous les coefficients sont rangés sur une seule et même ligne, ce que suggère la seconde initialisation (nous y reviendrons).

Comme pour les tableaux unidimensionnels, l'initialisation peut être incomplête (et dans ce cas, les coefficients non renseignés valent 0). La déclaration `int mat [2][4]={{1},{2,3}};` peut être représentée par la matrice :

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 2 & 3 & 0 & 0 \end{pmatrix}$$

.

On accède au coefficient ligne 1 colonne 2 par la syntaxe `mat[1][2]` :

— Exemple d'accès en lecture `printf("%d",mat[1][2]);` .

— Exemple d'accès en écriture `mat[1][2]=19;` .

Enfin, il est toujours possible de faire d'abord la déclaration puis de donner une valeur aux coefficients (par exemple au moyen d'une double boucle imbriquée).

Comme pour les tableaux unidimensionnels, aucune information sur les dimensions d'un tableau de tableaux n'est connue en dehors du bloc de déclaration. Il est sage de prévoir, dans les fonctions de manipulations de tableaux de tableaux, deux paramètres pour les dimensions.

Terminons par l'exemple d'une fonction qui somme les éléments d'une matrice :

```c
int somme(int n, int m, int t[n][m]){
  /*
  t : matrice de n x m entiers
  */
  int s = 0;
  int i = 0, j=0;
  while (i<n){
    j = 0;
    while (j<n){
      s = s + t[i][j];
      j++;
    }// fin while j
    i++;
  }// fin while i
  return s;
}
```

Exercices

Affichage de tableaux et matrices

On tente dans cette partie de bien aligner les colonnes des matrices affichées. À ce propos, on peut consulter avec profit cet échange sur Quora. On choisit d'afficher les flottants avec deux décimales.

Exercice 1. Affichage de tableaux unidimensionnels et bi-dimensionnels.

Q.1 Écrire la procédure `void display_line(int n, float t[n])` qui affiche un tableau de flottants.

Tests :

```c
void test_display_line(){
  float t[3] = {12.,13.256,100.659};
  display_line(3,t);
  printf("\n");
}

int main(void){
  test_display_line();
  //test_display_mat();
 return 0;
}
```

```
$ gcc -Wall dm1.c -o dm1
$ ./dm1
 12.00   13.26  100.66
```

Q.2 Écrire la procédure `void display_mat(int n, int m,float t[n][m])` qui affiche le contenu d'une matrice en alignant bien les colonnes.

Tests :

```c
void test_display_mat(){
  float t[3][3] = {{6.25,3.2,4.891},{12.,13.256},{1.123,256.3,568.}};
  display_mat(3,3,t);
  printf("\n");
}

int main(void){
  test_display_mat();
  return 0;
}
```

```
$ gcc -Wall dm1.c -o dm1
$ ./dm1
  6.25    3.20    4.89
 12.00   13.26    0.00
  1.12  256.30  568.00
```

Systèmes d'équations

Dans les exercices suivants, un système d'équation est représenté par une matrice (pour les coefficients du système) et un vecteur (pour les seconds membres).

Par exemple :

```c
float m[2][2] = {{1.,2.},{-1.,1.}};
float v[2] = {0.,1.};
```

représente le système

$$\begin{cases} x & + & 2y & = & 0 \\ -x & + & y & = & 1 \end{cases}$$

Exercice 2. Écrire une fonction `int solve2(int t[2][2], int v[2], int res[2]);` qui renvoie 1 si le système a une solution unique et 0 sinon. La solution unique éventuelle est écrite dans les coefficients du tableau `res`.

Notons que, comme les dimensions des tableaux sont ici connues, il n'est pas nécessaire de les passer en argument.

Tests :

```c
void test_solve2(void){
  float m[2][2] = {{1.,2.},{-1.,1.}};
  float v[2] = {0.,1.};
```

```
4     float r[2] = {0.,0.};
5     int nb = solve2(m,v,r);
6     if (nb){
7       printf("solution unique\n");
8       printf("%f,%f\n",r[0],r[1]);
9     }
10    else
11      printf("déterminant nul\n");
12  }
13
14  int main(void){
15    test_solve2();
16    //test_sarrus();
17    return 0;
18  }
19
```

```
$ ./dm1
solution unique
-0.666667,0.333333
```

Exercice 3. Dans cet exercice, toutes les matrices sont de dimensions 3×3 et les vecteurs sont des tableaux à 3 éléments.

Le *déterminant* d'une matrice est une valeur réelle qui indique si la matrice est *inversible* ou non (donc si un système d'équations formé par les coefficients de cette matrice a une solution unique ou non).

Remarque. Lorsque le déterminant de la matrice du système est nul, le système admet soit aucune soit une infinité de solutions.

Q.1 Un déterminant d'ordre 3 se calcule facilement à la main par les *règles de Sarrus* (cf. Wikipedia)

Écrire la fonction `float sarrus(float m[3][3])` qui calcule le déterminant d'une matrice carrée d'ordre 3.

Tests :

```
1  void test_sarrus(){
2    float m[3][3] = {{1,2,-1},{-1,0,1},{2,-1,2}};
3    printf("sarrus(m)=%f\n",sarrus(m));
4  }
```

```
$ ./dm1
sarrus(m)=8.000000
```

Q.2 La résolution d'un système d'équation peut se faire mécaniquement par les *règles de Cramer* (cf. Wikipedia).

Il faut en premier lieu calculer les matrices notées A_k dans l'article de Wikipedia. Il s'agit de remplacer la colonne k de la matrice du système par la matrice des seconds membres.

Écrire la fonction `void remplacer(float m[3][3],float v [3],int k)` qui remplace la colonne k de la matrice m par le vecteur v.

Tests :

```c
void test_remplacer(){
  float m[3][3] = {{1,2,-1},{-1,0,1},{2,-1,2}};
  float v[3] = {5.,5.,6.};
  remplacer(m,v,2);
  display_mat(3,3,m);
  printf("\n");
}

```

Avec un `main` convenable puis compilation et exécution :

```
$ gcc -Wall dm1.c -o dm1
$ ./dm1
  1.00    2.00    5.00
 -1.00    0.00    5.00
  2.00   -1.00    6.00
```

Affichage d'une lettre

Exercice 4. On veut afficher une grande lettre à l'écran.

Q.1 Écrire la fonction `void afficheN(int n, char c)` qui affiche à l'écran un « N » de hauteur n formé avec le caractère c.

Tests :

```c
void test_afficheN(){
  afficheN(6,'x');
  printf("\n");
}

```

Avec un `main` convenable puis compilation et exécution :

```
$ gcc -Wall dm1.c -o dm1
$ ./dm1
x     x
xx    x
x x   x
x  x x
x   xx
x     x
```

Q.2 Écrire la fonction `void afficheA(int n, char c)` qui affiche à l'écran un « A » de hauteur n formé avec le caractère c. La barre horizontale du A est au $2/5$ de sa hauteur.

Tests :

```c
void test_afficheA(){
  afficheA(8,'x');
  printf("\n");
}

```

Avec un `main` convenable puis compilation et exécution :

```
$ gcc -Wall dm1.c -o dm1
$ ./dm1
        x
       x x
      x   x
     xxxxxxx
    x       x
   x         x
  x           x
 x             x
```