Introduction à la programmation en C

Cours 3
23/01/2013

Tableaux (suite),
lecture/écriture de fichiers de texte .

Samy BLUSSEAU, Miguel COLOM

I. Les tableaux (suite).

I. Les tableaux (suite)

Les tableaux à 2 dimensions (et plus).

- Un tableau peut représenter différents objets, tels qu'un vecteur, une fonction d'une variable discrète (un signal) ...

- Certains objets nécessitent d'identifier leurs éléments par 2 indices (ou plus)

Exemple : Matrices, tableaux à double entrée, fonctions de deux variables discrètes (signaux de dimension 2, tels que les images)

Les tableaux à 2 dimensions (et plus) : *comment faire* ?

Méthode 1 : *Les tableaux de tableaux.*

Statiques

```
float t[2][2] ;

t[0][0] = 123 ;
t[0][1] = 3.1415 ;
t[1][0] = 0.07 ;
t[1][1] = 23.12013 ;
```

Les tableaux à 2 dimensions (et plus) : *comment faire* ?

Méthode 1 : *Les tableaux de tableaux (linked list).*

Dynamiques

```
int m ;
int n ;

float** t = (float**) malloc( m*sizeof( float* ) ) ;
for(i = 0; i <m; i++)
   t[i] = (float*)malloc(n*sizeof(float));

 for( i = 0 ; i < m; i++)
   for( j = 0; j < n; j++ )
     t[i][j] = (i+j)/2 ;

 for( i = 0 ; i < m; i++)
     free( t[i] ) ;

 free( t ) ;
```

Les tableaux à 2 dimensions (et plus) : *comment faire* ?

Méthode 1 : *Les tableaux de tableaux.*

Dans les deux cas,

- t est un float**,
- chaque t[i] est un float*, un tableau
- chaque t[i][j] est un float

Il n'y a pas de raison pour que l'adresse contenue dans t[i] soit voisine de l'adresse contenue dans t[i+1] .

Ainsi, **deux « lignes » du tableau peuvent se trouver à des endroits très différents en mémoire**

→ Fragmentation de la mémoire, lenteur...
mais surtout : **plus compliqué à écrire** (plusieurs *malloc*, plusieurs *free*...)

Exercice :

- Créer un fichier tableaux_2D.c.
- Dans le ***main()***, créer dynamiquement, par la méthode 1, un tableau de taille, type, et valeurs de votre choix.
- Afficher à l'écran la valeur et l'adresse de chaque entrée.
- Que remarque-t-on dans la suite des adresses mémoire?

Les tableaux à 2 dimensions (et plus) : *comment faire ?*

Méthode 2 : *Des tableaux « 1D » pour tout représenter !*

Dynamiques (Idem pour le cas statique, sans ***malloc*** !)

```
int m = ... ;
int n = ...;

float* t = (float*) malloc( m*n*sizeof( float ) ) ;

for( i = 0 ; i < m; i++)
   for( j = 0; j < n; j++ )
     t[n*i + j] = (i+j)/2 ;

free(t) ;
```

Les tableaux à 2 dimensions (et plus) : *comment faire* ?

Méthode 2 : *Des tableaux « 1D » pour tout représenter !*

Ce qui compte ici :

Méthode 1 **t[i] [j]** **<=>** **t[n*i + j]** *Méthode 2*

Mais **nous recommandons méthode 2** en général :

- plus simple (un seul ***malloc,*** un seul ***free*** !)
- universelle (un seul type de tableaux)

Exercice : Ecrire un programme dans un fichier sum_matrice.c, qui :

- Définit deux matrices de tailles et valeurs de votre choix .
  Ex :
  int M = 3 ;
  int N = 2 ;
  float matrice[M*N] = {1,2, 3,4 , 5,6} ;
  (ou des valeurs données par l'utilisateur...)

- Somme ces matrices et stocke le résultat dans une troisième

- Affiche le résultat

II. Les fichiers de texte.

II. Les fichiers

- En C il existe un type **FILE ***, qui représente un flux de données, notamment un fichier.
- On utilise les fonctions **fopen** et **fclose** de *stdio.h* pour ouvrir et fermer un fichier.
- On utilise souvent **fscanf** et **fprintf** pour lire et écrire du texte formaté.
- **fgets** : pour lire une ligne de texte.

Dans toutes ces fonctions, le **"f"** signifie **"file"**
fscanf et *fprintf* peuvent être changés en *sscanf* et *sprintf*, avec un "s" pour "string", chaîne de caractères.

fopen, fprintf, fclose

Exemple d'écriture avec fprintf :

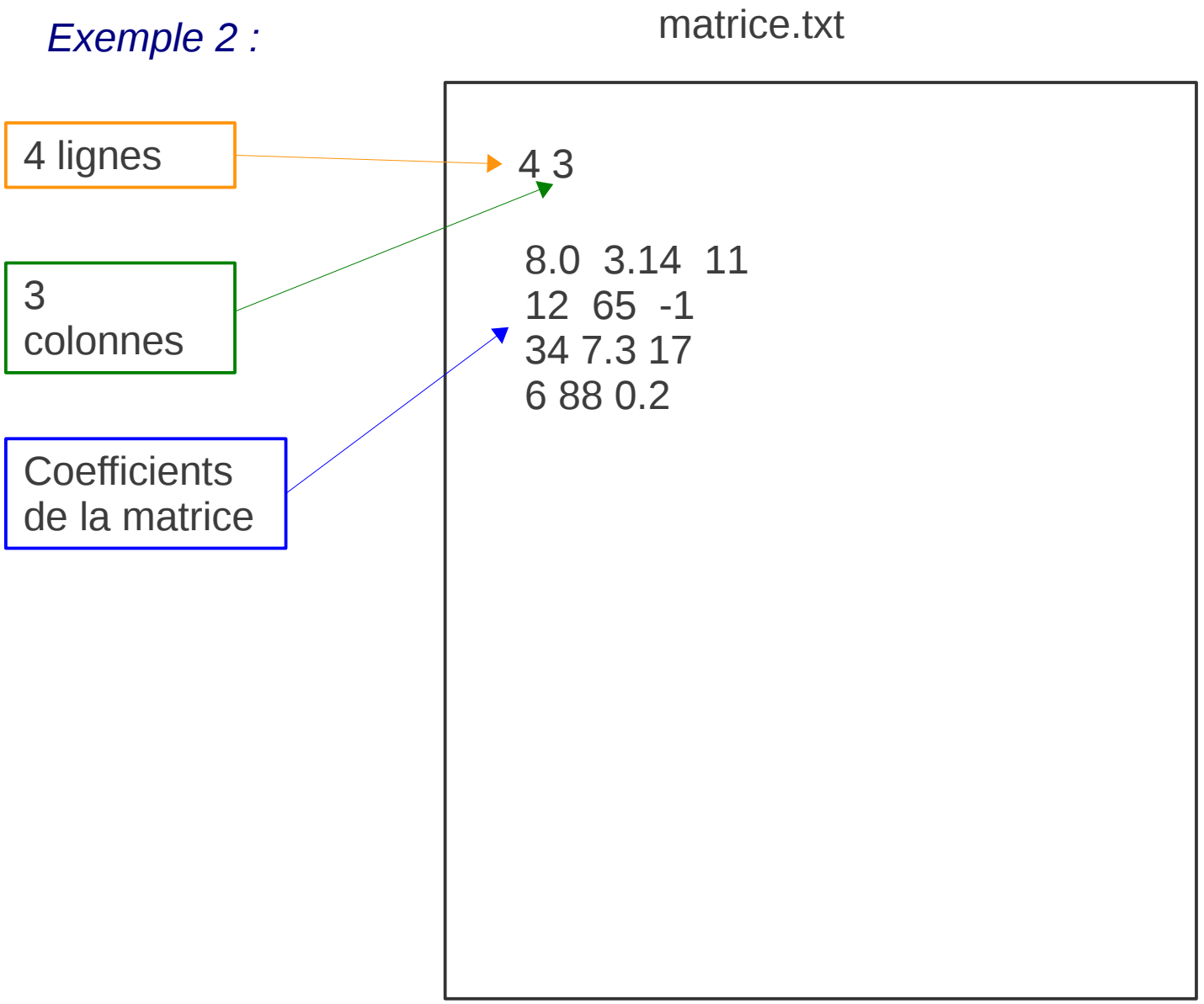
```
const char* nom_fichier =  "fichier1.txt" ;
FILE*  fic = fopen( nom_fichier , "w" );  /*ouverture en ecriture, "w" pour "write" */

fprintf( fic, "Bonjour!\n " ) ;

fclose( fic ) ;
```

Un fichier « **fichier1.txt** » a été **créé** dans le répertoire courant s'il n'existait pas,
ou a **écrasé** le précédent s'il en existait déjà un.
Il contient le texte "Bonjour !" .

Pour ne pas écraser un fichier déjà existant et écrire à la suite :
remplacer "w" par "a" (*append*).

fopen, fscanf, fclose

Exemple 2 :



fopen, fscanf, fclose

Exemple 2 (lecture avec fscanf):

```
char* nom_fichier =  "matrice.txt" ;
int n, m ;
float* M ;
FILE*  fic = fopen( nom_fichier , "r" );  /*ouverture en lecture, "r" pour "read" */

if( fic == NULL){
      printf("Impossible d'ouvrir le fichier %s\n", nom_fichier) ;
      exit( EXIT_FAILURE ) ;
}

fscanf( fic, "%d  %d " , &n, &m ) ;  /*Lecture des tailles, stockées en n et m */

M = (float*) malloc( n*m*sizeof(float) ) ; /* Allocation du tableau */
for(i=0;i<n;i++)
      for(j=0;j<m;j++) {
            fscanf( fic, "%f", &(M[m*i+j]) ); /*Lecture des coefficients, stockés dans M */
            printf("Read number %f at (%d,%d)\n", M[m*i+j],  i,j ) ;
      }
fclose( fic ) ;
```

fopen, fscanf, fclose

Contrôle d'erreur

fscanf retourne

- **un entier > 0 si** la conversion s'est bien déroulée
- **0** s'il y a eu erreur de conversion

  Ex :

```
float x ;
fscanf(fic, " %f", &x ) ;
/*Retourne 0 si la valeur lue n'est pas un float*/
```

- **-1 si** on est déjà arrivé à la fin du fichier

fopen, fscanf, fclose

Exemple 2 (avec contrôle d'erreur) :

```
char* nom_fichier =  "matrice.txt" ;
int n, m ;
float* M ;
FILE*  fic = fopen( nom_fichier , "r" );  /*ouverture en lecture, "r" pour "read" */

if( fic == NULL){
        printf("Impossible d'ouvrir le fichier %s\n", nom_fichier) ;
        exit( EXIT_FAILURE ) ;
}

int nread ;
nread = fscanf( fic, "%d  %d " , &n, &m ) ;  /*Lecture des tailles, stockées en n et m */
if( nread <= 0){
        fprintf( stderr, "Conversion Error!\n" ) ;
        exit(EXIT_FAILURE) ;
}
M = (float*) malloc( n*m*sizeof(float) ) ; /* Allocation du tableau */
for(i=0;i<n;i++)
        for(j=0;j<m;j++) {
                nread = fscanf( fic, "%f", &(M[m*i+j]) ); /*Lecture des coefficients, stockés dans M */
                if( nread <= 0){
                        fprintf( stderr, "Conversion error at (%d,%d)!\n",i,j ) ;
                        exit(EXIT_FAILURE) ;
                }
                printf("Read number %f at (%d,%d)\n", M[m*i+j],  i,j ) ;
        }
fclose( fic ) ;
```

fopen, fscanf, fclose

Exemple de lecture avec fgets et sscanf :

```
const char* nom_fichier =  "fichier1.txt" ;
const int MAX_LEN = 250 ;
char premier_mot[MAX_LEN] ; /*chaîne de caractères */
FILE*  fic = fopen( nom_fichier , "r" ) ;  /*ouverture en lecture, "r" pour "read" */

if( fic == NULL){
      printf("Impossible d'ouvrir le fichier %s\n", nom_fichier) ;
      exit( EXIT_FAILURE ) ;
}

fgets( premier_mot, MAX_LEN, fic ) ; /*lecture d'une ligne  */
sscanf( premier_mot, "%s " , premier_mot ) ; /*analyse de la chaîne et recopie*/

fclose( fic ) ;
```

Toujours s'assurer qu'un fichier a été correctement ouvert ! Dans le cas contraire, arrêter le programme.

Exercice : Créer une bibliothèque algebre_lineaire.c. qui permette de

- Lire et écrire des matrices dans des fichiers de texte
  Quelques exemples :
  FILE *f = read_matrix_size(int *m, int *n, char *filename) ;
  read_matrix(float *M, int m, int n, FILE *f) ;

- Sommer, soustraire, multiplier des matrices, et plus si vous le souhaitez !
  Ex :
  add_matrices(float *S, float *A, float *B) ;

Ne pas oublier l'en-tête correspondante (.h)!

Tester ces fonctions dans un fichier de votre choix, en ayant créé auparavant le Makefile correspondant.