Comment ‘bien’ déclarer et utiliser en C un tableau de dimension 2 (matrice)

```
#include <stdio.h>                  // pour printf, puts, putchar
#include <stdlib.h>                 // pour malloc, free

#define NDIM1  3                    // par exemple
#define NDIM2  5                    // par exemple


/*
Voici une routine d’affichage utilisable dans le cas d’un dimensionnement statique. En C, on peut seulement omettre la première dimension des tableaux c’est-à-dire qu’il faut impérativement que la fonction connaisse les autres dimensions. C’est pour cela que cette fonction n’est utilisable que dans le premier cas.
*/

void affiche_1 ( int T[][NDIM2] , int n1 , int n2 )
{
   int i, j ;

   putchar( ’\n’ ) ;
   for ( i = 0 ; i < n1 ; ++i )
   {
      for ( j = 0 ; j < n2 ; ++j )
         printf( " i = %3d ,  j = %3d   ===>  T1 = %5d \n",
                 i, j, T[i][j] ) ;
   }
   putchar( ’\n’ ) ;
}


/*
Voici une routine d’affichage utilisable dans le cas d’un dimensionnement dynamique d’un objet considéré comme une matrice.
*/

void affiche_2 ( int * * T , int n1 , int n2 )
{
   int i, j ;

   putchar( ’\n’ ) ;
   for ( i = 0 ; i < n1 ; ++i )
   {
      for ( j = 0 ; j < n2 ; ++j )
      printf( " i = %3d ,  j = %3d   ===>  T2 = %5d \n",
              i, j, T[i][j] ) ;
   }
   putchar( ’\n’ ) ;
}
```

```
/*
```
Voici une routine d'affichage utilisable dans le cas d'un dimensionnement dynamique d'un objet considéré comme un vecteur.
```
*/

void affiche_3 ( int * V , int n1 , int n2 )
{
   int i, j, k ;

   putchar( '\n' ) ;
   for ( i = 0 ; i < n1 ; ++i )
   {
      for ( j = 0 ; j < n2 ; ++j )
      {
          k = i * n2 + j ;              // calcul de l'adresse
          printf( " i = %3d ,  j = %3d ,  k = %3d   ===>  V = %5d \n",
                  i, j, k, V[k] ) ;
      }
   }
   putchar( '\n' ) ;
}


/*
```
Voici, après les routines d'affichage (placées en tête que pour des raisons de compilation), le programme principal avec les différentes possibilités et les explications qui vont avec.

Le cas '1' présente une allocation entièrement statique de la matrice. Dans le programme principal donné en illustration, le tableau déclaré ainsi aura comme nom `ANZ` .

Le cas '2' présente une allocation entièrement dynamique de la matrice i.e. chaque ligne '`T[i]`' est (réellement) indépendante des autres et donc il n'y a aucune raison que le stockage en mémoire des éléments de la matrice soit contigu (en revanche, chaque ligne peut ainsi être allouée avec une taille différente mais c'est une autre application que celle, ici, de déclarer une matrice). En ce qui concerne les matrices triangulaires ou en bande, même si cette solution est admissible, la solution présentée au cas suivant, moyennant bien sûr une adaptation, est bien meilleure. Dans le programme principal donné en illustration, le tableau déclaré ainsi aura comme nom `BP` .

Le cas '3' présente une allocation dynamique mais où on préserve la contiguïté des éléments. C'est donc la 'solution' préconisée et pour apprendre à s'en servir, on se reportera aux explications données dans le programme principal. Dans ce programme principal, tableau déclaré ainsi aura comme nom `PIC` .
```
*/


void main (void)
{
   int i, j, n1 = NDIM1 , n2 = NDIM2 ;

   int ANZ[NDIM1][NDIM2] ;             // dimensionnement statique
   int * * BP ;                        // dimensionnement dynamique
   int * * PIC ;                       // dimensionnement dynamique
```

```
/* CAS STATIQUE (lignes contiguës) */

   puts( " Cas 1 : ANZ mode \n" ) ;


/*
On regarde les tailles dans le cas du dimensionnement statique
*/

   printf( " sizeof de ANZ : %d \n", sizeof(ANZ) ) ;
   printf( " taille de ANZ : %d \n", sizeof(ANZ)/sizeof(**ANZ) ) ;
   putchar( '\n' ) ;


/*
On peut donc remplir de suite le tableau (par exemple, avec un incrément de 10 par ligne)
*/

   for ( i = 0 ; i < NDIM1 ; ++i )
   {
      for ( j = 0 ; j < NDIM2 ; ++j ) ANZ[i][j] = 10 * i + j ;
   }


/*
Et regarder ce qu'il y a dedans
*/

   affiche_1 ( ANZ , NDIM1 , NDIM2 ) ;
// affiche_2 ( (int * *)ANZ , n1 , n2 ) ;        // run time error ? !
   affiche_3 ( (int *)ANZ , n1 , n2 ) ;

   puts( " ============================================== \n" ) ;


/* CAS DYNAMIQUE (lignes non nécessairement contiguës) */

   puts( " Cas 2 : BP  mode \n" ) ;


/*
On alloue d'abord le tableau contenant les pointeurs des lignes (bien voir la taille demandée)
*/

   BP = (int * *) malloc( n1 * sizeof( int * ) ) ;
   if ( NULL == BP ) puts( " PB1 " ) ;


/*
Puis, on alloue chaque ligne à la longueur souhaitée mais on remarque que chaque ligne est bien
totalement indépendante des autres.
*/

   for ( i = 0 ; i < n1 ; ++i )
   {
      BP[i] = ( int * ) malloc( n2 * sizeof(int) ) ;
      if ( NULL == BP[i] ) printf( " PB2 : i = %d \n", i ) ;
   }
```

```
/*
On peut alors regarder les tailles, en fait maintenant celles des pointeurs.
*/

   printf( " sizeof de  BP : %d \n", sizeof(BP) ) ;
   printf( " sizeof de *BP : %d \n", sizeof(*BP) ) ;
   putchar( '\n' ) ;


/*
On peut donc alors remplir le tableau (par exemple, avec un incrément de 100 par ligne)
*/

   for ( i = 0 ; i < n1 ; ++i )
   {
      for ( j = 0 ; j < n2 ; ++j ) BP[i][j] = 100 * i + j ;
   }


/*
Et regarder de qu'il y a dedans. Selon les ordinateurs, on peut constater, que l'affichage par la
routine 'affiche_3' qui suppose que les éléments sont réellement contigus, peut donner n'importe
quoi.
*/

   affiche_2 ( BP , n1 , n2 ) ;
   affiche_3 ( *BP , n1 , n2 ) ;


/*
Et ne pas oublier, si l'on veut être propre de libérer la mémoire le plus tôt possible i.e. ne pas
attendre la fin du programme. On le fait, bien sûr, dans l'ordre inverse de celui de l'allocation.
*/

   for ( i = 0 ; i < n1 ; ++i ) free( BP[i] ) ;
   free( BP ) ;

   puts( " ============================================== \n" ) ;


/*  CAS DYNAMIQUE (lignes contiguës)  */

   puts( " Cas 3 : PIC  mode \n" ) ;


/*
On alloue d'abord le tableau contenant les pointeurs des lignes (bien voir la taille demandée).
Là, pas de changement par rapport au cas précédent.
*/

   PIC = (int * *) malloc( n1 * sizeof( int * ) ) ;
   if ( NULL == PIC ) puts( " PB3 " ) ;
```

```
/*
```
Puis, on alloue pour la première ligne la taille totale de la matrice (bien voir la taille demandée). Après, maintenant que la totalité de l'espace de stockage consacrée à la matrice a bien été allouée, on le partitionne selon les tailles souhaitées à chaque ligne (c'est à cet endroit que l'on peut introduire les modifications correspondantes aux cas des matrices triangulaires ou des matrices bande). On se sert pour cela des propriétés de l'arithmétique des pointeurs.
```
*/

   PIC[0] = (int *) malloc( n1 * n2 * sizeof(int) ) ;
   if ( NULL == PIC[0] ) puts( " PB4 " ) ;
   for ( i = 1 ; i < n1 ; ++i ) PIC[i] = PIC[i-1] + n2 ;


/*
On peut alors regarder les tailles, en fait maintenant celles des pointeurs.
*/

   printf( " sizeof de  PIC : %d \n", sizeof(PIC)  ) ;
   printf( " sizeof de *PIC : %d \n", sizeof(*PIC) ) ;
   putchar( '\n' ) ;


/*
On peut donc alors remplir le tableau (par exemple, avec un incrément de 1000 par ligne)
*/

   for ( i = 0 ; i < n1 ; ++i )
   {
      for ( j = 0 ; j < n2 ; ++j ) PIC[i][j] = 1000 * i + j ;
   }


/*
Et regarder ce qu'il y a dedans.
*/

   affiche_2 ( PIC , n1 , n2 ) ;
   affiche_3 ( PIC[0] , n1 , n2 ) ;
   puts( " ============================================== \n" ) ;


/*
Et ne pas oublier, si l'on veut être propre de libérer la mémoire le plus tôt possible i.e. ne pas attendre la fin du programme. On le fait, bien sûr, dans l'ordre inverse de celui de l'allocation.
*/

   free( *PIC ) ;
   free( PIC ) ;


   puts( " \n C'est fini ! \n" ) ;
}
```

Et voici les sorties de ce programme :

```
Cas 1 : ANZ mode

 sizeof de ANZ : 60
 taille de ANZ : 15

 i =   0 ,  j =   0   ===>  T1 =     0
 i =   0 ,  j =   1   ===>  T1 =     1
 i =   0 ,  j =   2   ===>  T1 =     2
 i =   0 ,  j =   3   ===>  T1 =     3
 i =   0 ,  j =   4   ===>  T1 =     4
 i =   1 ,  j =   0   ===>  T1 =    10
 i =   1 ,  j =   1   ===>  T1 =    11
 i =   1 ,  j =   2   ===>  T1 =    12
 i =   1 ,  j =   3   ===>  T1 =    13
 i =   1 ,  j =   4   ===>  T1 =    14
 i =   2 ,  j =   0   ===>  T1 =    20
 i =   2 ,  j =   1   ===>  T1 =    21
 i =   2 ,  j =   2   ===>  T1 =    22
 i =   2 ,  j =   3   ===>  T1 =    23
 i =   2 ,  j =   4   ===>  T1 =    24

 i =   0 ,  j =   0 ,  k =   0   ===>  V =     0
 i =   0 ,  j =   1 ,  k =   1   ===>  V =     1
 i =   0 ,  j =   2 ,  k =   2   ===>  V =     2
 i =   0 ,  j =   3 ,  k =   3   ===>  V =     3
 i =   0 ,  j =   4 ,  k =   4   ===>  V =     4
 i =   1 ,  j =   0 ,  k =   5   ===>  V =    10
 i =   1 ,  j =   1 ,  k =   6   ===>  V =    11
 i =   1 ,  j =   2 ,  k =   7   ===>  V =    12
 i =   1 ,  j =   3 ,  k =   8   ===>  V =    13
 i =   1 ,  j =   4 ,  k =   9   ===>  V =    14
 i =   2 ,  j =   0 ,  k =  10   ===>  V =    20
 i =   2 ,  j =   1 ,  k =  11   ===>  V =    21
 i =   2 ,  j =   2 ,  k =  12   ===>  V =    22
 i =   2 ,  j =   3 ,  k =  13   ===>  V =    23
 i =   2 ,  j =   4 ,  k =  14   ===>  V =    24

 ============================================

 Cas 2 : BP  mode

 sizeof de  BP : 4
 sizeof de *BP : 4

 i =   0 ,  j =   0   ===>  T2 =     0
 i =   0 ,  j =   1   ===>  T2 =     1
 i =   0 ,  j =   2   ===>  T2 =     2
 i =   0 ,  j =   3   ===>  T2 =     3
 i =   0 ,  j =   4   ===>  T2 =     4
 i =   1 ,  j =   0   ===>  T2 =   100
 i =   1 ,  j =   1   ===>  T2 =   101
 i =   1 ,  j =   2   ===>  T2 =   102
 i =   1 ,  j =   3   ===>  T2 =   103
 i =   1 ,  j =   4   ===>  T2 =   104
 i =   2 ,  j =   0   ===>  T2 =   200
 i =   2 ,  j =   1   ===>  T2 =   201
```

```
i =   2 ,  j =   2   ===>  T2 =   202
i =   2 ,  j =   3   ===>  T2 =   203
i =   2 ,  j =   4   ===>  T2 =   204

i =   0 ,  j =   0 ,  k =   0   ===>  V =     0
i =   0 ,  j =   1 ,  k =   1   ===>  V =     1
i =   0 ,  j =   2 ,  k =   2   ===>  V =     2
i =   0 ,  j =   3 ,  k =   3   ===>  V =     3
i =   0 ,  j =   4 ,  k =   4   ===>  V =     4
i =   1 ,  j =   0 ,  k =   5   ===>  V = -33686019
i =   1 ,  j =   1 ,  k =   6   ===>  V = 524296
i =   1 ,  j =   2 ,  k =   7   ===>  V = 524544
i =   1 ,  j =   3 ,  k =   8   ===>  V = 3082168
i =   1 ,  j =   4 ,  k =   9   ===>  V = 3082296
i =   2 ,  j =   0 ,  k =  10   ===>  V =     0
i =   2 ,  j =   1 ,  k =  11   ===>  V =     0
i =   2 ,  j =   2 ,  k =  12   ===>  V =    20
i =   2 ,  j =   3 ,  k =  13   ===>  V =     1
i =   2 ,  j =   4 ,  k =  14   ===>  V =    46

================================================

Cas 3 : PIC  mode

sizeof de  PIC : 4
sizeof de *PIC : 4

i =   0 ,  j =   0   ===>  T2 =     0
i =   0 ,  j =   1   ===>  T2 =     1
i =   0 ,  j =   2   ===>  T2 =     2
i =   0 ,  j =   3   ===>  T2 =     3
i =   0 ,  j =   4   ===>  T2 =     4
i =   1 ,  j =   0   ===>  T2 =  1000
i =   1 ,  j =   1   ===>  T2 =  1001
i =   1 ,  j =   2   ===>  T2 =  1002
i =   1 ,  j =   3   ===>  T2 =  1003
i =   1 ,  j =   4   ===>  T2 =  1004
i =   2 ,  j =   0   ===>  T2 =  2000
i =   2 ,  j =   1   ===>  T2 =  2001
i =   2 ,  j =   2   ===>  T2 =  2002
i =   2 ,  j =   3   ===>  T2 =  2003
i =   2 ,  j =   4   ===>  T2 =  2004

i =   0 ,  j =   0 ,  k =   0   ===>  V =     0
i =   0 ,  j =   1 ,  k =   1   ===>  V =     1
i =   0 ,  j =   2 ,  k =   2   ===>  V =     2
i =   0 ,  j =   3 ,  k =   3   ===>  V =     3
i =   0 ,  j =   4 ,  k =   4   ===>  V =     4
i =   1 ,  j =   0 ,  k =   5   ===>  V =  1000
i =   1 ,  j =   1 ,  k =   6   ===>  V =  1001
i =   1 ,  j =   2 ,  k =   7   ===>  V =  1002
i =   1 ,  j =   3 ,  k =   8   ===>  V =  1003
i =   1 ,  j =   4 ,  k =   9   ===>  V =  1004
i =   2 ,  j =   0 ,  k =  10   ===>  V =  2000
i =   2 ,  j =   1 ,  k =  11   ===>  V =  2001
i =   2 ,  j =   2 ,  k =  12   ===>  V =  2002
i =   2 ,  j =   3 ,  k =  13   ===>  V =  2003
i =   2 ,  j =   4 ,  k =  14   ===>  V =  2004

================================================
```

```
C'est fini !
```