Introduction au langage C

13 janvier 2010

Table des matières

Chapitre 1

Les tableaux et les structures

1.1 Les tableaux

1.1.1 Les tableaux à une dimension

1.1.1.1 Définition

Un tableau à une dimension ou uni dimensionnel `A` est une variable structurée formée d'un nombre entier `N` de variables simples du même type, qui sont appelées les composantes du tableau. Le nombre de composantes `N` est alors la dimension du tableau.

En faisant le rapprochement avec les mathématiques, on dit encore que `A` est un vecteur de dimension `N`.

Exemple :

La déclaration : `int jours[12]=31,28,31,30,31,30,31,31,30,31,30,31 ;`

définit un tableau de type `int` de `dimension 12`. Les `12 composantes` sont initialisées par les valeurs respectives `31, 28, 31, ... , 31`. On peut accéder à la première composante du tableau par `jours[0]`, à la deuxième composante par `jours[1]`, ..., à la dernière composante par `jours [11]`.

1.1.1.2 Déclaration et mémorisation

Déclaration

- Déclaration de tableaux en langage algorithmique :
  `<TypeSimple> tableau <NomTableau>[<Dimension>]`
- Déclaration de tableaux en C :
  `<TypeSimple> <NomTableau>[<Dimension>] ;`

Les noms des tableaux sont des identificateurs qui doivent correspondre aux restrictions définies au chapitre 1.

Exemples :

Les déclarations suivantes en langage algorithmique :

```
entier tableau A[25]
réel tableau B[100]
booléen tableau C[10]
caractère tableau D[30]
```

se traduisent en langage C par :

```
int A[25]; ou bien long A[25]; ou bien ...
float B[100]; ou bien double B[100]; ou bien ...
int C[10];
char D[30];
```

Mémorisation

En langage C, le nom d'un tableau est le représentant de l'adresse du premier élément du tableau. Les adresses des autres composantes sont calculées automatiquement relativement à cette adresse.

Exemple :

```
short A[5] = 1200, 2300, 3400, 4500, 5600 ;
```

Si un tableau est formé de `N` composantes et si une composante a besoin de `M octets` en mémoire, alors le tableau occupera de `N*M octets`.

Exemple :

En supposant qu'une variable du type `long` occupe `4 octets`, (c'est à dire : `sizeof(long)=4)`), pour le tableau `T` déclaré par : `long T[15]`, le langage C réservera `N*M = 15*4 = 60 octets` en mémoire.

1.1.1.3 Initialisation et réservation automatique

Initialisation

Lors de la déclaration d'un tableau, on peut initialiser les composantes du tableau, en indiquant la liste des valeurs respectives entre accolades.

Exemples :

```
int A[5]   = {10, 20, 30, 40, 50};
float B[4] = {-1.05, 3.33, 87e-5, -12.3E4};
int C[10]  = {1, 0, 0, 1, 1, 1, 0, 1, 0, 1};
```

Il faut évidemment veiller à ce que le nombre de valeurs dans la liste corresponde à la dimension du tableau. Si la liste ne contient pas assez de valeurs pour toutes les composantes, les composantes restantes sont initialisées par zéro.

Réservation automatique

Si la dimension n'est pas indiquée explicitement lors de l'initialisation, alors l'ordinateur réserve automatiquement le nombre d'octets nécessaires.

Exemples :

```
int A[] = {10, 20, 30, 40, 50};
      // réservation de 5*sizeof(int) octets (dans notre cas: 10 octets)
float B[] = {-1.05, 3.33, 87e-5, -12.3E4};
      // réservation de 4*sizeof(float) octets (dans notre cas: 16 octets)
int C[] = {1, 0, 0, 1, 1, 1, 0, 1, 0, 1};
      // réservation de 10*sizeof(int) octets (dans notre cas: 20 octets)
```

1.1.1.4 Accès aux composantes

En déclarant un tableau par : `int A[5] ;` nous avons défini un tableau `A avec cinq composantes`, auxquelles on peut accéder par : `A[0], A[1], ... , A[4]`.

Exemples :

1. `MAX = (A[0]>A[1]) ? A[0] : A[1];`
   `A[4] *= 2;`
2. Considérons un tableau `t` de dimension `n` :

   a) En langage algorithmique,
   – l'accès au premier élément du tableau se fait par t[1] ;

– l'accès au dernier élément du tableau se fait par t[n].

b) En langage C,
– l'accès au premier élément du tableau se fait par `t[0]` ;
– l'accès au dernier élément du tableau se fait par `t[n-1]`.

1.1.1.5 Affichage et affectation

La structure `for` se prête particulièrement bien au travail avec les tableaux. La plupart des applications se laissent implémenter par simple modification des exemples types de l'affichage et de l'affectation.

Affichage du contenu d'un tableau

Traduisons le programme `Afficher` du langage algorithmique en langage C :

Langage algorithmique

```
programme Afficher
|   entier tableau A[5]
|   entier I  (* Compteur *)
|   pour I variant de 1 à 5 faire
|     écrire A[I]
|   fpour
fprogramme
```

Langage C

```
main()
{
      int A[5];
      int I;  /* Compteur */
      for (I=0; I<5; I++)
      printf("%d ", A[I]);
      return 0;
      printf("\n");
}
```

Remarques :

1. Avant de pouvoir afficher les composantes d'un tableau, il faut évidemment leur affecter des valeurs ;
2. Rappelez-vous que la deuxième condition dans la structure `for` n'est pas une condition d'arrêt, mais une condition de répétition ! Ainsi la commande d'affichage sera répétée aussi longtemps que `I` est inférieur à `5`. La boucle sera donc bien exécutée pour les indices `0,1,2,3 et 4` !
3. Par opposition à la commande simplifiée `ecrire A[I]` du langage algorithmique, la commande `printf()` doit être informée du type exact des données à afficher. (Ici : %d ou %i pour des valeurs du type `int`) ;
4. Pour être sûr que les valeurs sont bien séparées lors de l'affichage, il faut inclure au moins un espace dans la chaîne de format. Autres possibilités :
   – printf("%d‸", A[I]) ; /* tabulateur */
   – printf("%7d", A[I]) ; /* format d'affichage */

Affectation : Affectation avec des valeurs provenant de l'extérieur

Traduisons le programme `Remplir` du langage algorithmique en langage C :

Langage algorithmique

```
programme REMPLIR
|   entier tableau A[5]
|   entier I  (* Compteur *)
|   pour I variant de 1 à 5 faire
|     lire A[I]
|   fpour
fprogramme
```

Langage C

```
main()
{
 int A[5];
 int I;  /* Compteur */
 for (I=0; I<5; I++)
     scanf("%d", &A[I]);
 return 0;
}
```

Remarques :

- Comme `scanf()` a besoin des adresses des différentes composantes du tableau, il faut faire précéder le terme texttt`A[I]` par l'opérateur adresse `&` ;
- La commande de lecture `scanf()` doit être informée du type exact des données à lire. (Ici : %d ou %i pour lire des valeurs du type `int`).

Exercices :

1. Ecrire un programme qui lit la dimension `N` d'un tableau `T` du type `int` (dimension maximale : 50 composantes), remplit le tableau par des valeurs entrées au clavier et affiche le tableau. Calculer et afficher ensuite la somme des éléments du tableau ;
2. Ecrire un programme qui lit la dimension `N` d'un tableau `T` du type int (dimension maximale : 50 composantes), remplit le tableau par des valeurs entrées au clavier et affiche le tableau. Effacer ensuite toutes les occurrences de la valeur `0` dans le tableau `T` et tasser les éléments restants. Afficher le tableau résultant ;
3. Ecrire un programme qui lit la dimension `N` d'un tableau `T` du type `int` (dimension maximale : 50 composantes), remplit le tableau par des valeurs entrées au clavier et affiche le tableau. Ranger ensuite les éléments du tableau `T` dans l'ordre inverse sans utiliser de tableau d'aide. Afficher le tableau résultant. *Idée : échanger les éléments du tableau à l'aide de deux indices qui parcourent le tableau en commençant respectivement au début et à la fin du tableau et qui se rencontrent en son milieu* ;
4. Ecrire un programme qui lit la dimension `N` d'un tableau `T` du type int (dimension maximale : 50 composantes), remplit le tableau par des valeurs entrées au clavier et affiche le tableau ; copiez ensuite toutes les composantes strictement positives dans un deuxième tableau `TPOS` et toutes les valeurs strictement négatives dans un troisième tableau `TNEG`. Afficher les tableaux `TPOS` et `TNEG` ;

5. *Produit scalaire de deux vecteurs :* écrire un programme qui calcule le produit scalaire de deux vecteurs d'entiers `U` et `V` (de même dimension) :

   Exemple :

   ```
   /          \   /          \
   | 3  2  -4 | * | 2  -3  5 | = 3*2+2*(-3)+(-4)*5 = -20
   \          /   \          /
   ```

6. *Maximum et minimum des valeurs d'un tableau :* écrire un programme qui détermine la plus grande et la plus petite valeur dans un tableau d'entiers `A`. Afficher ensuite la valeur et la position du `maximum` et du `minimum`. Si le tableau contient plusieurs `maxima` ou `minima`, le programme retiendra la position du premier `maximum` ou `minimum` rencontré ;
7. *Insérer une valeur dans un tableau trié :* un tableau `A` de dimension `N+1` contient `N` valeurs entières triées par ordre croissant ; la `(N+1) ième` valeur est indéfinie. Insérer une valeur `VAL` donnée au clavier dans le tableau `A` de manière à obtenir un tableau de `N+1` valeurs triées ;
8. *Recherche d'une valeur dans un tableau :* rechercher dans un tableau d'entiers `A` une valeur `VAL` entrée au clavier. Afficher la position de `VAL` si elle se trouve dans le tableau, sinon afficher un message correspondant. La valeur `POS` qui est utilisée pour mémoriser la position de la valeur dans le tableau, aura la valeur `-1` aussi longtemps que `VAL` n'a pas été trouvée. Implémenter deux versions :

   a) La recherche séquentielle : comparer successivement les valeurs du tableau avec la valeur donnée ;

   b) La recherche dichotomique ('recherche binaire', 'binary search'). *Condition : le tableau `A` doit être trié* ;
      - Comparer le nombre recherché à la valeur au milieu du tableau,
        - s'il y a égalité ou si le tableau est épuisé, arrêter le traitement avec un message ; correspondant ;
        - si la valeur recherchée précède la valeur actuelle du tableau, continuer la recherche dans le demi tableau à gauche de la position actuelle ;
        - si la valeur recherchée suit la valeur actuelle du tableau, continuer la recherche dans le demi tableau à droite de la position actuelle.
      - Ecrire le programme pour le cas où le tableau `A` est trié par ordre croissant. Quel est l'avantage de la recherche dichotomique ? Expliquer brièvement.

9. *Fusion de deux tableaux triés :*
   - Problème : on dispose de deux tableaux `A` et `B` (de dimensions respectives `N` et `M`), triés par ordre croissant. Fusionner les éléments de `A` et `B` dans un troisième tableau `FUS` trié par ordre croissant ;
   - Méthode : utiliser trois indices `IA`, `IB` et `IFUS`. Comparer `A[IA]` et `B[IB]` ; remplacer `FUS[IFUS]` par le plus petit des deux éléments ; avancer dans le tableau `FUS` et dans le tableau qui a contribué son élément. Lorsque l'un des deux tableaux `A` ou `B` est épuisé, il suffit de recopier les éléments restants de l'autre tableau dans le tableau `FUS` ;

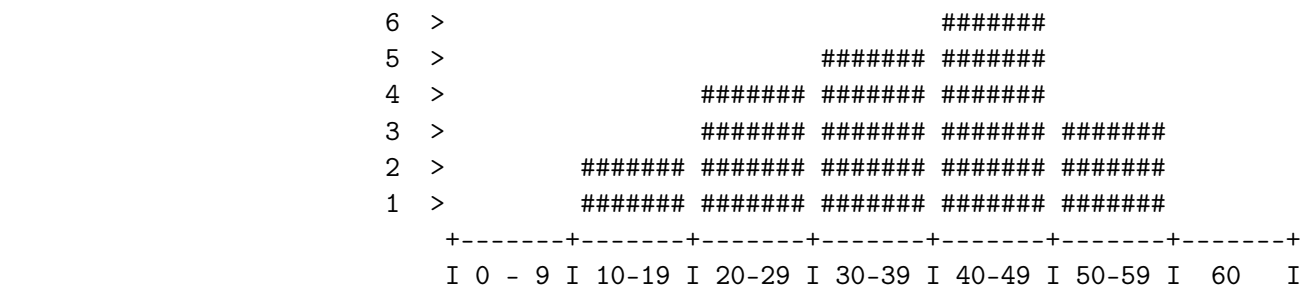
10. *Tri par sélection du maximum :*
    - Problème : Classer les éléments d'un tableau `A` par ordre décroissant ;
    - Méthode : Parcourir le tableau de gauche à droite à l'aide de l'indice `I`. Pour chaque élément `A[I]` du tableau, déterminer la position `PMAX` du (premier) maximum à droite de `A[I]` et échanger `A[I]` et `A[PMAX]` ;
11. *Tri par propagation (bubble sort)*
    - Problème : Classer les éléments d'un tableau `A` par ordre croissant ;
    - Méthode : En recommençant chaque fois au début du tableau, on effectue à plusieurs reprises le traitement suivant : on propage, par permutations successives, le plus grand élément du tableau vers la fin du tableau (comme une bulle qui remonte à la surface d'un liquide) ;
    - Exemple : Implémenter l'algorithme en considérant que :
      - La partie du tableau (à droite) où il n'y a pas eu de permutations est triée ;
      - Si aucune permutation n'a eu lieu, le tableau est trié.
12. *Statistique des notes :* écrire un programme qui lit les points de `N` élèves d'une classe dans un devoir et les mémorise dans un tableau `Points` de dimension `N` :
    (a) Rechercher et afficher :
        - la note maximale,
        - la note minimale,
        - la moyenne des notes.

    (b) A partir des points des élèves, établir un tableau `Notes` de dimension `7` qui est composé de la façon suivante :
        - `Notes` [6] contient le nombre de notes 60 ;
        - `Notes` [5] contient le nombre de notes de 50 à 59 ;
        - `Notes` [4] contient le nombre de notes de 40 à 49 ;
        - ... ;
        - `Notes` [0] contient le nombre de notes de 0 à 9.

        Etablir un graphique de barreaux représentant le tableau `Notes`. Utilisez les symboles `#######` pour la représentation des barreaux et affichez le domaine des notes en dessous du graphique.
        - Idée : Déterminer la valeur maximale `MAXN` dans le tableau `Notes` et afficher autant de lignes sur l'écran. (Dans l'exemple ci-dessous, `MAXN = 6`).

Exemple :

- La note maximale est 58
- La note minimale est 13
- La moyenne des notes est 37.250000



1.1.2 Les tableaux à deux dimensions

1.1.2.1 Définitions

En langage C, un tableau à deux dimensions `A` est à interpréter comme un tableau (uni dimensionnel) de dimension `L` dont chaque composante est un tableau, uni dimensionnel, de dimension K. On appelle `L` le nombre de lignes du tableau et `K` le nombre de colonnes du tableau. `L` et `K` sont alors les deux dimensions du tableau. Un tableau à deux dimensions contient donc `L*K` composantes.

On dit qu'un tableau à deux dimensions est `carré`, si `L` est égal à `K`. En faisant le rapprochement avec les mathématiques, on peut dire que `A est un vecteur de L vecteurs de dimension K`, ou mieux, `A est une matrice de dimensions L et K`.

Exemple :

Considérons un tableau `Note` à une dimension pour mémoriser les notes de 20 élèves d'une classe dans un devoir :

```
int Note[20] = 45, 34, ... , 50, 48 ;
```

Pour mémoriser les notes des élèves dans les 10 devoirs d'un trimestre, nous pouvons rassembler plusieurs de ces tableaux uni dimensionnels dans un tableau `Note` à deux dimensions :

```
int Note [10][20] = {{45, 34, ... , 50, 48},
{39, 24, ... , 49, 45},
...    ...     ...
{40, 40, ... , 54, 44}};
```

Dans une ligne nous retrouvons les notes de tous les élèves dans un devoir. Dans une colonne, nous retrouvons toutes les notes d'un élève.

1.1.2.2 Déclaration et mémorisation

Déclaration

- Déclaration de tableaux à deux dimensions en langage algorithmique :
  `<TypeSimple> tableau <NomTabl>[<DimLigne>,<DimCol>]`
- Déclaration de tableaux à deux dimensions en langage C :
  `<TypeSimple> <NomTabl>[<DimLigne>][<DimCol>] ;`

Exemples :

Langage algorithmique	Langage C
`Les déclarations suivantes en langage algorithmique:`	`se traduisent en langage C par:`
`entier tableau A[10,10]`	`int A[10][10]; ou long A[10][10]; ou ...`
`réel tableau B[2,20]`	`float B[2][20]; ou double B[2][20]; ou ...`
`booléen tableau C[3,3]`	`int C[3][3];`
`caractère tableau D[15,40]`	`char D[15][40];`

Mémorisation

Comme pour les tableaux à une dimension, le nom d'un tableau est le représentant de l'adresse du premier élément du tableau (c'est à dire l'adresse de la première ligne du tableau). Les composantes d'un tableau à deux dimensions sont stockées ligne par ligne dans la mémoire.

Exemple :

Mémorisation d'un tableau à deux dimensions :

```
short A[3][2] = {{1, 2},
        {10,  20 },
        {100, 200}};
```

Un tableau de dimensions `L` et `K`, formé de composantes dont chacune a besoin de `M octets`, occupera `L*K*M octets` en mémoire.

Exemple :

En supposant qu'une variable du type `double` occupe `8 octets` (c'est à dire : `sizeof(double)=8`), pour le `tableau T` déclaré par : `double T[10][15]`, le langage C réservera `L*K*M =`

`10*15*8 = 1200 octets` en mémoire.

1.1.2.3 Initialisation et réservation automatique

Initialisation

Lors de la déclaration d'un tableau, on peut initialiser les composantes du tableau, en indiquant la liste des valeurs respectives entre accolades. A l'intérieur de la liste, les composantes de chaque ligne du tableau sont encore une fois comprises entre accolades. Pour améliorer la lisibilité des programmes, on peut indiquer les composantes dans plusieurs lignes.

Exemples :

```
int A[3][10] = {{0,10,20,30,40,50,60,70,80,90},
                {10,11,12,13,14,15,16,17,18,19},
                {1,12,23,34,45,56,67,78,89,90}};

float B[3][2] = {{-1.05, -1.10},
                 {86e-5, 87e-5},
                 {-12.5E4, -12.3E4}};
```

Lors de l'initialisation, les valeurs sont affectées ligne par ligne en passant de gauche à droite. Nous ne devons pas nécessairement indiquer toutes les valeurs : Les valeurs manquantes seront initialisées par zéro. Il est cependant défendu d'indiquer trop de valeurs pour un tableau.

Réservation automatique

Si le nombre de lignes L n'est pas indiqué explicitement lors de l'initialisation, l'ordinateur réserve automatiquement le nombre d'octets nécessaires.

Exemples :

```
int A[][10] = {{ 0,10,20,30,40,50,60,70,80,90},
               {10,11,12,13,14,15,16,17,18,19},
               { 1,12,23,34,45,56,67,78,89,90}};
//réservation de 3*10*2 = 60 octets

float B[][2] = {{-1.05,   -1.10  },
                {86e-5,    87e-5 },
                {-12.5E4, -12.3E4}};
//réservation de 3*2*4 = 24 octets
```

1.1.2.4 Accès aux composantes

– Accès à un tableau à deux dimensions en langage Algorithmique :
`<NomTableau>[<Ligne>, <Colonne>]`
– Accès à un tableau à deux dimensions en langage C :
`<NomTableau>[<Ligne>][<Colonne>]`

Les éléments d'un tableau de dimensions `L` et `K` se présentent de la façon suivante :

```
/                                                          \
|  A[0][0]     A[0][1]     A[0][2]     . . .     A[0][C-1]  |
|  A[1][0]     A[1][1]     A[1][2]     . . .     A[1][C-1]  |
|  A[2][0]     A[2][1]     A[2][2]     . . .     A[2][C-1]  |
|    . . .       . . .       . . .     . . .       . . .    |
| A[L-1][0]   A[L-1][1]   A[L-1][2]    . . .   A[L-1][C-1]  |
\                                                          /
```

Considérons un tableau `A` de dimensions `L` et `K` :

En langage algorithmique,
– les indices du tableau varient de `1` à `L`, respectivement de `1` à `K` ;
– la composante de la `Nième` ligne et `Mième` colonne est notée : `A[N,M]` ;
En langage C,

– les indices du tableau varient de `0` à `L-1`, respectivement de `0` à `L-1` ;
– la composante de la `Nième` ligne et Mième colonne est notée : `A[N-1][M-1]` ;

1.1.2.5 Affichage et affectation

Lors du travail avec les tableaux à deux dimensions, nous utiliserons deux indices (par exemple : I et J), et la structure `for`, souvent imbriquée, pour parcourir les lignes et les colonnes des tableaux.

Affichage du contenu d'un tableau à deux dimensions

Traduisons le programme `Afficher` du langage algorithmique en langage C :

Langage algorithmique

```
Algorithmique

programme AFFICHER
|   entier tableau A[5,10]
|   entier I,J
|   (* Pour chaque ligne ... *)
|   pour I variant de 1 à 5 faire
|     (* ... considérer chaque
  composante *)
|     pour J variant de 1 à 10 faire
|       écrire A[I,J]
|     fpour
|     (* Retour à la ligne *)
|     écrire
|   fpour
fprogramme
```

Langage C

```
Langage C
main()
{
 int A[5][10];
 int I,J;
 /* Pour chaque ligne ... */
 for (I=0; I<5; I++)
   {
    /* ... considérer chaque composante */
    for (J=0; J<10; J++)
   printf("%7d", A[I][J]);
    /* Retour à la ligne */
    printf("\n");
   }
 return 0;
}
```

Remarques :

- Avant de pouvoir afficher les composantes d'un tableau, il faut leur affecter des valeurs.
- Pour obtenir des colonnes bien alignées lors de l'affichage, il est pratique d'indiquer la largeur minimale de l'affichage dans la chaîne de format. Pour afficher des matrices du type int (valeur la plus 'longue' : -32768), nous pouvons utiliser la chaîne de format "%7d" : printf("%7d", A[I][J]).

Affectation avec des valeurs provenant de l'extérieur

Traduisons le programme `Remplir` du langage algorithmique en langage C :

Langage algorithmique

```
Algorithmique
programme REMPLIR
|   entier tableau A[5,10]
|   entier I,J
|   (* Pour chaque ligne ... *)
|   pour I variant de 1 à 5 faire
|     (* ... considérer chaque
    composante *)
|     pour J variant de 1 à 10 faire
|       lire A[I,J]
|     fpour
|   fpour
fprogramme
```

Langage C

```
Langage C
main()
{
 int A[5][10];
 int I,J;
 /* Pour chaque ligne ... */
 for (I=0; I<5; I++)
   /* ... considérer chaque composante */
   for (J=0; J<10; J++)
   scanf("%d", &A[I][J]);
 return 0;
}
```

Exercices :

1. Ecrire un programme qui lit les dimensions `L` et `K` d'un tableau `T` à deux dimensions du type `int` (dimensions maximales : 50 lignes et 50 colonnes). Remplir le tableau par des valeurs entrées au clavier et afficher le tableau ainsi que la somme de tous ses éléments ;
2. Ecrire un programme qui lit les dimensions `L` et `K` d'un tableau `T` à deux dimensions du type `int`(dimensions maximales : 50 lignes et 50 colonnes). Remplir le tableau par des valeurs entrées au clavier et afficher le tableau ainsi que la somme de chaque ligne et de chaque colonne en n'utilisant qu'une variable d'aide pour la somme ;
3. Ecrire un programme qui transfère un tableau `M` à deux dimensions `L` et `K` (dimensions maximales : 10 lignes et 10 colonnes) dans un tableau `V` à une dimension `L*K` :

   Exemple :

```
/         \
| a b c d |        /                         \
| e f g h |  ==>   | a b c d e f g h i j k l |
| i j k l |        \                         /
\         /
```

4. *Mise à zéro de la diagonale principale d'une matrice :* écrire un programme qui met à zéro les éléments de la diagonale principale d'une matrice carrée A donnée ;
5. *Matrice unitaire :* écrire un programme qui construit et affiche une matrice carrée unitaire `U` de dimension `N`. Une matrice unitaire est une matrice, telle que :

```
/ 1 si i=j
uij =

|

\ 0 si ij
```

6. *Transposition d'une matrice :* écrire un programme qui effectue la transposition `tA` d'une matrice `A` de dimensions `N` et `M` en une matrice de dimensions `M` et `N` :
   a) La matrice transposée sera mémorisée dans une deuxième matrice `B` qui sera ensuite affichée.
   b) La matrice `A` sera transposée par permutation des éléments.

```
Rappel:
/         \      /       \
tA =  t | a b c d |  =   | a e i |
| e f g h |      | b f j |
| i j k l |      | c g k |
\         /      | d h l |
 \       /
```

7. *Multiplication d'une matrice par un réel :* écrire un programme qui réalise la multiplication d'une matrice `A` par un réel `X` :

```
Rappel:

/         \   /                  \
| a b c d |   | X*a X*b X*c X*d |
    X * | e f g h | = | X*e X*f X*g X*h |
| i j k l |   | X*i X*j X*k X*l |
        \         /   \                 /
```

   – Le résultat de la multiplication sera mémorisé dans une deuxième matrice A qui sera ensuite affichée.
   – Les éléments de la matrice A seront multipliés par X.

8. *Addition de deux matrices :* écrire un programme qui réalise l'addition de deux matrices A et B de mêmes dimensions N et M :

   `Rappel:`

```
/         \   /             \     /                       \
| a b c d |   | a' b' c' d' |     | a+a' b+b' c+c' d+d' |
| e f g h | + | e' f' g' h' |  =  | e+e' f+f' g+g' h+h' |
| i j k l |   | i' j' k' l' |     | i+i' j+j' k+k' l+l' |
\         /   \             /     \                       /
```

   – Le résultat de l'addition sera mémorisé dans une troisième matrice `C` qui sera ensuite affichée ;
   – La matrice `B` est ajoutée à `A`.

9. *Triangle de Pascal :* écrire un programme qui construit le triangle de PASCAL de degré `N` et le mémorise dans une matrice carrée `P` de dimension `N+1` :

   Exemple : Triangle de Pascal de degré 6 :

```
n=0   1
n=1   1 1
n=2   1 2 1
n=3   1 3 3 1
n=4   1 4 6 4 1
n=5   1 5 10 10 5 1
n=6   1 6 15 20 15 6 1
```

   Méthode :

   Calculer et afficher seulement les valeurs jusqu'à la diagonale principale (incluse). Limiter le degré à entrer par l'utilisateur à 13. Construire le triangle ligne par ligne :

   – Initialiser le premier élément et l'élément de la diagonale à 1 ;
   – Calculer les valeurs entre les éléments initialisés de gauche à droite en utilisant la relation : `Pi,j = Pi-1,j + Pi-1,j-1` ;

10. *Recherche de* `points-cols` *:*
    Rechercher dans une matrice donnée A les éléments qui sont à la fois un maximum sur leur ligne et un minimum sur leur colonne. Ces éléments sont appelés des points-cols. Afficher les positions et les valeurs de tous les points-cols trouvés :

    Exemple : les éléments soulignés sont des `points cols` :

```
/           \   /         \   /           \   /       \
| 1 8 3 4 0 |   | 4 5 8 9 |   | 3 5 6 7 7 |   | 1 2 3 |
|           |   | 3 8 9 3 |   | 4 2 2 8 9 |   | 4 5 6 |
| 6 7 2 7 0 |   | 3 4 9 3 |   | 6 3 2 9 7 |   | 7 8 9 |
\           /   \         /   \           /   \       /
```

    Méthode :

    Etablir deux matrices d'aide `MAX` et `MIN` de même dimensions que `A`, telles que :

```
 / 1 si A[i,j] est un maximum
MAX[i,j] = | sur la ligne i
 \ 0 sinon
  / 1 si A[i,j] est un minimum
```

1.2 Les structures

1.2.1 Différence entre une structure et un tableau

Un tableau permet de regrouper des éléments de même type, c'est-à-dire codés sur le même nombre de bits et de la même façon. Toutefois, il est généralement utile de pouvoir rassembler des éléments de type différent tels que des entiers et des chaînes de caractères. Les structures permettent de remédier à cette lacune des tableaux, en regroupant des objets (des variables) au sein d'une entité repérée par un seul nom de variable. Les objets contenus dans la structure sont appelés champs de la structure.

1.2.2 Déclaration d'une structure

Lors de la déclaration de la structure, on indique les champs de la structure, c'est à dire le type et le nom des variables qui la composent :

```
struct Nom_Structure
{
     type_champ1 Nom_Champ1;
     type_champ2 Nom_Champ2;
     type_champ3 Nom_Champ3;
     type_champ4 Nom_Champ4;
     type_champ5 Nom_Champ5;

     ...
};
```

Remarques :

1. la dernière accolade doit être suivie d'un point-virgule ;
2. le nom des champs répond aux critères des noms de variable ;
3. deux champs ne peuvent avoir le même nom ;
4. les données peuvent être de n'importe quel type hormis le type de la structure dans laquelle elles se trouvent.

Ainsi, la structure suivante est correcte :

```
struct MaStructure
{
     int Age;
     char Sexe;
     char Nom[12];
     float MoyenneScolaire;
     struct AutreStructure StructBis; /* en considérant que la structure
                     AutreStructure est définie */
};
```

Par contre la structure suivante est incorrecte :

```
struct MaStructure
{
     int Age;
     char Age;
     struct MaStructure StructBis;
};
```

Il y a deux raisons à cela :

1. le nom de variable Age n'est pas unique ;
2. le type de donnée struct MaStructure n'est pas autorisé.

La déclaration d'une structure ne fait que donner l'allure de la structure, c'est-à-dire en quelque sorte une définition d'un type de variable complexe. La déclaration ne réserve donc pas d'espace mémoire pour une variable structurée (variable de type structure), il faut donc définir une (ou plusieurs) variable(s) structurée(s) après avoir déclaré la structure...

1.2.3 Définition d'une variable structurée

La définition d'une variable structurée est une opération qui consiste à créer une variable ayant comme type celui d'une structure que l'on a précédemment déclarée, c'est à dire la nommer et lui réserver un emplacement en mémoire.

La définition d'une variable structurée se fait comme suit :

```
struct Nom_Structure Nom_Variable_Structuree;
```

- `Nom_Structure` représente le nom d'une structure que l'on aura préalablement déclarée ;
- `Nom_Variable_Structuree` est le nom que l'on donne à la variable structurée.

Il va de soi que, comme dans le cas des variables on peut définir plusieurs variables structurées en les séparant avec des virgules :

```
struct Nom_Structure Nom1, Nom2, Nom3, ...;
```

Exemple :

Soit la structure `Personne` suivante :

```
struct Personne
{
     int Age;
     char Sexe;

};
```

On peut définir plusieurs variables structurées : `struct Personne Fatou, Cheick-Omar, Habibou`.

Accès aux champs d'une variable structurée

Chaque variable de type structure possède des champs repérés avec des noms uniques. Toutefois le nom des champs ne suffit pas pour y accéder étant donné qu'ils n'ont de contexte qu'au sein de la variable structurée... Pour accéder aux champs d'une structure on utilise l'opérateur de champ (un simple point) placé entre le nom de la variable structurée que l'on a défini et le nom du champ :

```
Nom_Variable.Nom_Champ;
```

Ainsi, pour affecter des valeurs à la variable `Fatou` (variable de type struct Personne définie précédemment), on pourra écrire :

```
Fatou.Age = 22;
Fatou.Sexe = 'F';
```

1.2.4 Tableaux de structures

Etant donné qu'une structure est composée d'éléments de taille fixe, il est possible de créer un tableau ne contenant que des éléments du type d'une structure donnée. Il suffit de créer un tableau dont le type est celui de la structure et de le repérer par un nom de variable :

```
struct Nom_Structure Nom_Tableau[Nb_Elements];
```

Chaque élément du tableau représente alors une structure du type que l'on a défini...

Le tableau suivant (nommé Repertoire) pourra par exemple contenir 8 variables structurées de type struct `Personne` :

```
struct Personne Repertoire[8];
```

De la même façon, il est possible de manipuler des structures dans les fonctions.