Chap III : Les tableaux

Dans cette partie, on va étudier quelques structures de données de base tels que :

- **Les tableaux (vecteur et matrice).**
- Les chaînes de caractères.
- …

LA STRUCTURE DE TABLEAU

- **Introduction :** supposons qu'on ait une dizaine de variables de même type (entier par exemple) qu'on appellera A1, A2, A3, A4, … , A10.
  Ces dix variables simples occupent chacune un emplacement physique en mémoire, après leur déclaration.
  Il y a une autre possibilité : Les rassembler toutes dans une seule structure appelée « tableau » dans dix emplacements contigus mais avec un seul nom.

  Cela ressemble à cette structure.

A1	A2	A3	A4	A5	A6	A7	A8	A9	A10

  Qui représente les dix cases contigus A1, A2, A3, …, A10
  Sous le même nom : Le tableau A par exemple.
- Comme vous pouvez le remarquer, pour définir un tableau on doit donner son nom, son type et sa taille (nombre de cases).
  En langage C, la syntaxe est la suivante :

  Type-case nom-tableau[nombre-case] ;

  Attention : le nombre de cases est entre crochets et non entre parenthèses.

- Reprenons l'exemple ci-dessus, si on veut déclarer les dix variables on peut le faire de deux manières : L'une les déclarer une par une et l'autre les déclarer dans la même structure.

```
/* Déclaration une par une*/
main()
{
        Int A1 ;
        Int A2 ;
        .
        .
        .
        Int A10 ;
}
```

```
/*Déclaration dans une seule structure*/
main()
{
        Int A[10] ;

}
```

Faites un parallèle entre les deux déclarations et dites quelle est la meilleur des deux !
Maintenant qu'on ait une petite idée sur la structure tableau essayons d'approfondir nos connaissances.

DEFINITION D'UNE VARIABLE TABLEAU

- Dans le cours d'algorithmique, on a étudié deux genres de tableaux : les vecteurs (tableaux à une dimension) et les matrices (tableaux à deux dimensions), on va faire de même avec le langage C et mettre en évidences les quelques différences qui existent dans la gestion des tableaux entre le langage C et l'algorithmique.
- Le langage C propose un type tableau qui est plus contraignant qu'en algorithmique et même quelques langages de haut niveau. En effet, les indices de parcours des cases ne peuvent commencer qu'à zéro et non pas par une valeur quelconque. D'autre part, les indices sont obligatoirement numériques ou de type énuméré qu'on étudiera par la suite.
  Enfin, la gestion de tableaux de plusieurs dimensions est délicate car très proche des mécanismes sous-adjacents (modes d'adressage et allocation mémoire qui sera étudier ultérieurement).
- La définition d'un tableau à une dimension (vecteur) suit la syntaxe suivante :

```
Type-case       nom-tableau[nombre-case] ;
```

- **Exemple de bilan**

En langage C	En algorithmique variable
Char nom[30] ;	nom : vecteur de 30 caractère ;
Int A[10] ;	A : vecteur de 10 entier ;
Float note[5] ;	note : vecteur de 5 réel ;

- Il faut faire attention, les cases du vecteur sont numérotées de zéro à nombre-case – 1. D'autre part le langage n'assure aucun contrôle sur les indices de parcours employés, ils peuvent être négatif ou supérieurs à la taille indiquée (nombre-case), le seul résultat à craindre est un « bug » voire un plantage du programme !
- Un tableau à deux dimensions (matrice) est défini selon la syntaxe suivante :

Type-case nom-tableau[nbrligne][nbrcolonne] ;

Il faut remarquer que les deux nombres de cases (ligne et colonne) sont donnés séparément.

- **Exemple de bilan**

En langage C	En algorithmique variable
float note[10][3] ;	note : matrice de 10*3 réel ;

INITIALISATION DES CASES D'UN TABLEAU

- Pour initialiser un tableau (vecteur ou matrice) comme d'ailleurs une variable simple, on a le choix entre l'initialisation au même temps que la définition ou après à l'aide d'affectations ou même d'opérations d'input : c'est-à-dire que définition et initialisation sont faites séparément.

Regardons la syntaxe de l'initialisation :

- Pour le vecteur

  Type nom[nombre] = {V0, V1, … Vn} ;

- Pour la matrice

```
Type    nom[Nligne][Ncolonne]= {
                                {V00, V01, V02, …V0m},
                                {V1O, V11, V12, …V1m},
                                .
                                .
                                .
                                {Vn0, Vn1, Vn2, …Vnm}
                               } ;
```

  Pour cet exemple Nligne = n et Ncolonne = m.

- **<u>Exemple de bilan</u>**

```
main()
{
        Int tab[3] = {21, 9, 55} ;
        float  pts[4][3] = {    {1.0 , 0.0, 0.0},
                                {0.0, 2.0, 0.0},
                                {0.0, 0.0, 1.0},
                                {1.1, 1.1, 1.1}
                           } ;
}
```

- On peut définir (créer) un type tableau sans pour autant définir une variable par la syntaxe suivante :

  Typedef type-case nom-type[nombre-case] ;

- **Exemple**

En langage C	En algorithmique
	Déclaration
	type
Typedef char typnom[30] ;	typnom = vecteur de 30 caractères ;
	variable
Typnom nom ;	nom : typnom ;

Avant d'étudier le mécanisme de manipulation des tableaux (vecteur et matrice), on va se familiariser avec les chaînes de caractères.

CHAINES DE CARACTERES

- Le langage C n'offre pas un traitement confortable des chaînes de caractères. En effet, elles sont gérées sous forme d'un vecteur de taille fixe, dans lequel on ajoute une sentinelle au bout du contenu utilisable. Cette sentinelle est le caractère de code ASCII nul ('\o').
- Déclaration d'une chaine de caractères : Il suffit de définir un vecteur de caractères. Seulement, il faut tenir compte de la sentinelle et par conséquent la réservation d'une case en plus de la longueur utile est primordiale !

  La syntaxe est :

  Char nom-chaine[longueurutile + 1] ;

- **Exemple :**

En langage C	En algorithmique
	Déclaration
	Variable
Char nom[31] ;	nom : chaine de 30 caractère ;

- On peut aussi déclarer et initialiser en même temps une chaîne de caractères, selon la syntaxe suivant :

  Char nom-chaine[longueurutile + 1] = ''contenu'' ;

- **Exemple :**
  #include <string.h>
  Char ligne[81] = ''salem'' ;
  Char texte[81] = {'s', 'a', 'l', 'e', 'm', '\o'} ;

NB : Les deux initialisations sont équivalentes car en langage C, les chaînes de caractères sont considérées comme des tableaux de type caractère. Mais je préfère la première écriture qui est simple et nous donne l'impression qu'on est en train d'utiliser une chaîne de caractères et non un vecteur.

- Lorsqu'on manipule des chaînes de caractères, il est nécessaire de prendre en considération leurs représentations en mémoire. En particulier, les comparaisons de chaînes peuvent sembler surprenantes, c'est-à-dire il faut comparer les contenus et non les contenants qui sont considérés comme des adresses.
- **Exemple :**
  Char chaine1[81] = ''salem'' ;
  Char chaine2[81] = ''salem'' ;

  Si vous comparez les deux chaînes (chaine1 et chaine2), vous allez dire qu'elles sont égales, et bien non ! En langage C elles sont différentes car ce langage établit une distinction entre le contenu (les caractères : 's', 'a', 'l', 'e', 'm', '\o') et les contenants (chaine1 et chaine2) qui sont des adresses de leurs emplacements en mémoire et donc chaine1 ≠ chaine2, quoiqu'elles contiennent la même information !

- Un autre problème, peut surgir lorsqu'on affecte des chaînes. Pour illustrer cela, regardons l'exemple suivant :
  char chaine1[81] = ''bonjour'' ;
  char chaîne2[81] ;
  chaine2 = chaine1 ; /*Affectation d'une chaîne dans une autre*/

  L'affectation vous parer correcte et pourtant elle provoque une erreur de compilation !

- Opérations sur les chaînes de caractères : Si un programme va utiliser des chaînes de caractères, il est utile d'inclure le fichier d'entête **<string.h>** pour pouvoir profiter de certaines fonctions prédéfinies qui manipulent les chaînes telles que :
  - strlen() donne le nombre d'octets occupés par la chaîne.
  - strcpy() copie une chaîne dans une autre.
  - strupr() convertit la chaine en majuscule.
  - strcat() concatène une chaîne à une autre.
  - Etc …

  Il y a plusieurs fonctions de ce genre qui seront très utiles.

TRAITEMENT DES TABLEAUX

- Pour accéder à une case d'un tableau, il faut un indice de parcours pour les vecteurs et deux autres pour les matrices, le premier pour les lignes et le deuxième pour les colonnes.

  Exemple :

En langage C	En algorithmique
X = vect[i] ;	x ← vect(i) ;
Y = mat[i][j];	y ← mat(I, j);

- La lecture d'un vecteur est tributaire d'une boucle sur l'indice de parcours (la boucle for est souhaitable).
  Exemple : lecture du vecteur vect de N entier.

En langage C	En algorithmique

For (i=0; i<N ; i=i+1)
  Scanf(''%d\n'', &vect[i]) ;
	Pour i allant de 1 à N faire
	  lire(vect(i));
	Finpour

Attention: Dans ce cas la lecture de chaque case se fait sur une ligne, si on veut les lire toutes sur la même ligne, il suffit d'écrire scanf(''%d'', &vect[i]) ; c'est-à-dire se débarrasser du retour chariot \n.

- L'écriture d'un vecteur, idem que pour la lecture, une boucle est plus que nécessaire !

En langage C	En algorithmique

For (i=0; i<N ; i=i+1)
  printf(''%d\n'', &vect[i]) ;
	Pour i allant de 1 à N faire
	  ecrire(vect(i));
	Finpour

/*ou bien printf(''%d'', &vect[i]);*/

- Pour lire une matrice, deux boucles imbriquées sont nécessaires l'une sur les lignes et l'autre sur les colonnes.

Exemple : Lecture d'une matrice mat de type réel, de N lignes et M colonnes.

En Langage C	En algorithmique

For (i=0 ; i<N ; i=i+1)
{
  For (j=0 ; i<N ; i=i+1)
  scanf(''%f'', &mat[i][j]) ;
  printf(\n) ;
}
	pour i allant de 1 à N faire
	  pour j allant de 1 à M faire
	  lire(mat(i, j)) ;
	  finpour
	finpour

Dans cet exemple, la lecture se fait ligne par ligne. Voir le retour chariot.

- Pour écrire une matrice on utilisera aussi deux boucles imbriquées.

Exemple : Ecriture de la même matrice citée ci-dessus.

En Langage C	En algorithmique

For (i=0 ; i<N ; i=i+1)
{
  For (j=0; i<N ; i=i+1)
  printf(''%f'', &mat[i][j]) ;
  printf(\n) ;
}
	pour i allant de 1 à N faire
	  pour j allant de 1 à M faire
	  Ecriture(mat(i, j)) ;
	  finpour
	finpour

BIBLIOGRAPHIE

1. Langage C : Cours et références
   Pierre NERZIC Mars 2003

2. Langage C : Support de cours
   Patrick CORDE Mars 2006

3. Programmation en langage C
   M.C. BELAID Mars 2008

4. Algorithmique, structures de données et langage C
   J.M. ENJALBERT Janvier 2005

5. Langage C
   Claude Delannoy 2iéme édition 2009