Les tableaux

Les types simples et les types structurés

Le type simple

Une variable d'un type simple n'a qu'une valeur dans un moment précis. Toutes les types examinées jusqu'au ce moment sont simples.

Le type structuré

Une variable d'un type structuré a plusieurs valeurs au même moment.

Le tableau

Le tableau est un type structuré dont toutes les valeurs (appelées éléments) sont du même type.

La structure

Les éléments (champs) de la structure peuvent être de types différents et sont caractérisés par leurs noms.

Tableaux unidimensionnels

Définition : Structure (objet) qui a plusieurs éléments du même type. Chaque élément est accédé par le nom du tableau et une valeur d'indice qui détermine sa place dans le tableau. (fig. 7.1)

Indice	0	1	2	3	4	5	6	7	8	9
Valeurs	11	1	102	13	56	25	18	54	23	5

Fig. 7.1

En C l'indice du premier élément a toujours la valeur 0. C'est une conséquence de la réalisation des tableaux en ce langage.

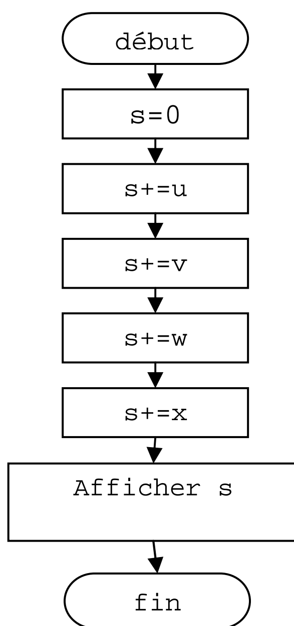


Fig. 7.2

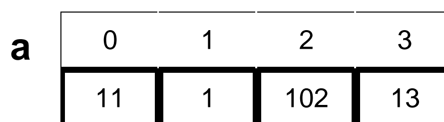


Fig. 7.3

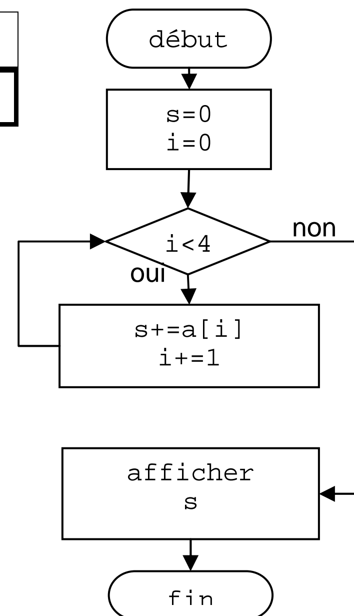


Fig. 7.4

Utilisation des tableaux

On utilise les tableaux quand on a plusieurs objets avec des propriétés similaires et on doit les traiter de même manière.

Exemple

Sommer les valeurs de quatre variables. On peut voir qu'on a besoin de minimum 4 instructions similaires(fig. 7.2). Mais si on utilise un tableau *a* de quatre éléments (fig.7.2) on peut utiliser le même nom et de ne changer que la valeur de l'indice qui est al valeur de la variable *i*. dans une boucle (fig.7.4). On voit bien que le corps de la boucle ne comporte que le l'algorithme de traitement de l'élément individuel et l'augmentation de la valeur de l'indice.

De cet exemple on peut déduire

L'algorithme générique de traitement d'un tableau

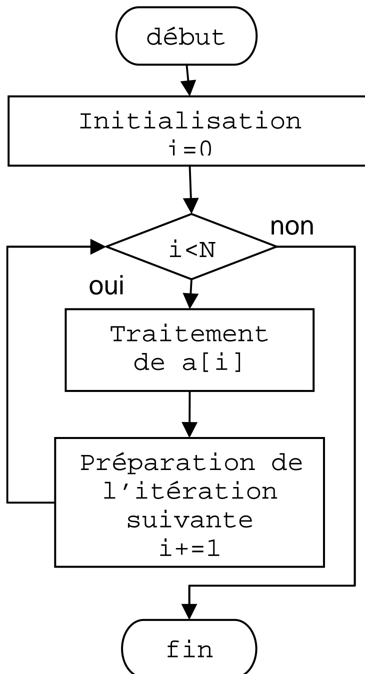


Fig. 7.5

Cet algorithme est montré à figure 7.5. Il comporte deux parties essentielles : initialisation où on doit au moins d'affecter une valeur initiale à la variable qui serve pour indice et la boucle elle-même. Dans la boucle on ne doit jamais oublier d'écrire les instructions nécessaires pour modifier l'indice.

Les tableaux unidimensionnels en C

Déclaration

Déclaration simple

Cette déclaration annonce le nom, le type de base (le type d'éléments) et la taille (le nombre d'éléments) du tableau. La taille doit être une expression constante parce que elle détermine le volume de la mémoire qui est engagé par le tableau. Elle ne détermine pas les valeurs des éléments. En C l'indice du premier élément est égal à 0 et l'indice du premier élément est égal à *taille – 1*.

```
type nom [taille];
```

Exemple :

Au dessous sont déclarés deus tableaux a'éléments entiers et un tableau d'éléments réels

```
int a[4], b[100];
float z[30];
```

Déclaration avec initialisation

La taille du tableau est déterminée par le nombre de valeurs de l'initialisation. Toutes les valeurs doivent être des **expressions constantes.**

```
type  nom []= [val1,val2,val3,...];
```

Exemple :

Un tableau d'entiers de quatre éléments

```
int a[] = {3,56,12,24};
```

Déclaration mixte

Dans ce cas la taille est déterminée la constante *taille* et le nombre de valeurs de l'initialisation doit être inférieur ou égal à lui. Quand le nombre de constantes d'initialisation est inférieur à *taille* les derniers éléments sont initialisés par 0.

```
type  nom [taille]= [val1,val2,val3,...];
```

Exemple :

Le tableau de réels *b* qui a 2 éléments est initialisé par les constantes *1.5* et *2.3*. Le tableau *c* est initialisé par les valeurs *1,2,3,0,0*

Réalisation des tableaux en C – arithmétique des pointeurs

En fait le nom du tableau c'est le nom d'un pointeur constant qui désigne le premier élément du tableau, c'est-à-dire la valeur du pointeur est l'adresse du premier élément (fig.7.6). En fait on peut écrire

a ≡ &a[0] ou *a ≡ a[0]

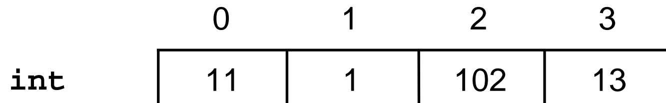


Fig. 7.6

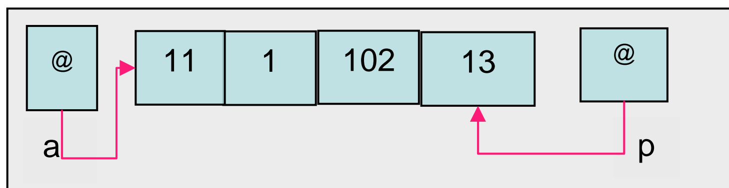


Fig. 7.7

La valeur d'un pointeur est l'adresse de l'objet désigné par lui. Mais l'adresse est une valeur entière et en C on peut ajouter ou soustraire des entiers pour obtenir de nouvelles adresses. L'ensemble de ces opérations sur les pointeurs est appelé l'arithmétique de pointeurs.

Additionner un entier à un pointeur (une adresse).

Le résultat est une adresse de la mémoire qui est l'adresse d'un objet du type de base du pointeur qui est à une distance égale à l'entier additionné de l'adresse initiale. C'est à dire que on additionne le produit de l'entier et la taille du type de base à la valeur du pointeur initial.

p+i en effet on ajout vers la valeur de **p** la valeur **i*sizeof(type of p)**

D'autre côté la différence entre deux pointeurs (toujours du même type) donne une valeur entière qui est égale au nombre d'objets du type de base qui se peuvent être placés entre les deus adresses.

Exemple (fig 7.6, 7.7):

```
int a[] = {11,1,102,3};
int *p = a+3;
printf("%d %d %d %d", *a, a[1],*(a+1),*(p-1));
printf(" ("%d %d %d %d ",a[2],*p,a[3],p-a )
```

Les expressions suivants sont vraies

```
a+1 == &a[1]
a+2 == &a[2]
```

et les résultats affichés seront

11 1 1 102

102 13 13 3

L'algorithme générique du traitement du tableau en C

L'algorithme de fig.7.5 peut être réalisé en C par la boucle

for :

```
for (i=0; i<N; i++){
   //Traitement de a[i]
}
```

ou ***while***

```
i=0
while (i<N){
   //Traitement de a[i]
   i++;
}
```

Exemple

Sommer les éléments d'un tableau.. On peut réaliser plusieurs variants. Le plus simple est d'utiliser la boucle for et sa variable de contrôle sera l'indice de l'élément traité.

```
#include <stdio.h>
void main (void) {
  int a[]= {11,3,6,34};
  int i,s = 0;
  for (i=0; i<4;i++) s+=a[i];  // ou s+=*(a+i)
  printf ("%d\n", s);
}
```

Le deuxième variant utilise comme variable du contrôle un pointeur, dont la valeur est l'adresse de l'élément traité.

```
#include <stdio.h>
void main (void) {
  int a[]= {11,3,6,34};
  int s = 0;
  int *p;
  for (p=a ; p-a<4;p++) s+=*p; // ou for (p=a ; p-a<4; ) s+=*p++;
  printf ("%d\n", s);
}
```

Le tableau comme paramètre d'une fonction

Paramètre formel

En fait le nom du tableau est un pointeur constant qui contient l'adresse du premier élément. De cette définition on peut conclure que le paramètre formel peut être déclaré soit comme un tableau sans spécification de nombre d'éléments

```
type nom_de_fonction(type nom[],...)
```

.

soit comme un pointeur , qui peut être *constant*

```
type nom_de_fonction(type *nom,...) ou

type nom_de_fonction(type *const nom,...)
```

Parce que le nombre d’éléments à traiter n’est pas spécifié il est indispensable d’ajouter toujours un paramètre de type entier qui spécifie le nombre d’éléments qui doivent être traités.

```
type nom_de_fonction(type nom[],int nombre,...)
```

Paramètre effectif

Le paramètre affectif fourni au temps d’appel doit être le nom du tableau réel quid doit être traité. Son type de base et le nombre de dimensions doivent correspondre avec le ceux du paramètre formel.

```
nom_de_fonction(nom_du_tableau,...)
```

Exemples

1. Au dessous on peut voir un petit programme qui trouve la somme des éléments d’un tableau de quatre éléments.

```
#include <stdio.h>
int intsomme(int a[], int n);
void main (void) {
  int a[]= {11,3,6,34};
  printf ("%d\n", intsomme(a,4));
}

int intsomme(int a[], int n){
  int i,s = 0;

for (i=0; i<n;i++) s+=a[i];
  return s;
}
```

L'appel de la fonction

Résultat: la somme des éléments

2 –ème paramètre: le nombre d'éléments qui doivent être sommés

1 –er paramètre: le tableau (l'adresse de son premier élément

L'instruction qui rend le résultat

On peut écrire le titre de la fonction *intsomme* dans 2 autres présences :

```
int intsomme(int *a, int n);
int intsomme(int *const a, int n);
```

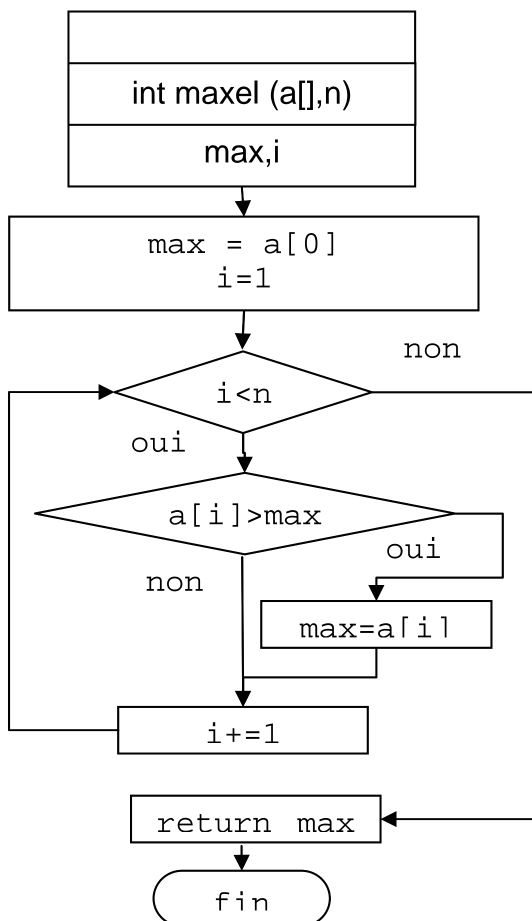
2 Ecrire deux fonctions :

- La première lit les éléments d’un tableau an lisant auparavant le nombre d’éléments qui est rendu comme le résultat de la fonction. Le deuxième paramètre est le nombre maximal d’éléments (la dimension du tableau)
- La deuxième fonction affiche les premiers N éléments d’un tableau. N est le deuxième paramètre.

```
int lir_tab( int a[], const int maxnom){
  int n,i;
  n = lit_entier_verifie(1,maxnom);
  if (n <0) return 0;
  printf("Tapez %d nombres entiers\n",n);
  for (i=0; i<n; i++)
        scanf("%d",&a[i]);
  return n;
}
void aff_tab( const int a[], const int n){
  int i;
  printf("Les %d éléments du tableau\n",n);
  for (i=0; i<n; i++)
        printf("%d, ",a[i]);
  printf ("\n");
}
```

3 Ecrire une fonction qui trouve la valeur maximale des éléments d'un tableau.

L'algorithme et le programme sont montrés à Fig. 7.8. A chaque répétition un élément est testé et si sa valeur est supérieure au maximum courant cette valeur devienne le nouveau maximum, Dan le cas contraire le valeur du maximum reste inchangée. Mais dans les deux cas la valeur du maximum est valide pour tous les éléments traite jusqu'au moment.




```
int lir_tab( int a[], const int maxnom){
  int n,i;
  n = lit_entier_verifie(1,maxnom);
  if (n <0) return 0;
  printf("Tapez %d nombres entiers\n",n);
  for (i=0; i<n; i++)
        scanf("%d",&a[i]);
  return n;
}
void aff_tab( const int a[], const int n){
  int i;
  printf("Les %d éléments du tableau\n",n);
  for (i=0; i<n; i++)
        printf("%d, ",a[i]);
  printf ("\n");
}
```


Fig. 7.8

Tableaux bidimensionnels

Définition et déclaration

Quand les éléments d'un tableau sont des tableaux eux-même bous parlons pour des tableaux bidimensionnels. Ce nom vient du fait que chaque élément est accédé à l'aide de deux indices. En C il y a plusieurs manières d'implémentation de ce consept. Le premier est le tableau rectangulaire (Fig. 7.9). On présent se tableau comme

une matrice dont chaque ligne est un tableau unidimensionnel. Pour chaque éléemnt le premier indice est l'indice de la ligne et le deuxième – celui de la colonne.

m	0	1	2	3
m[0]:	12	5	43	11
m[1]:	3	23	84	12
m[2]:	5	7	9	10

m[1][3]

Fig. 7.9

La déclaration d'un tableau bidimensionnel rassemble à celle d'un tableau unidimensionnel mais on doit spécifier les tailles des deux dimensions.

```
type nom [nombre_lignes] [nombre_colonnes];
```

Exemple

```
int m[3][4];
```

Disposition dans la mémoire

Le tableau est situé linéairement dans la mémoire parce que la mémoire elle-même est linéaire. Les lignes sont placées une après l'autre (fig. 7.10).

m[0]	1	2	3	m[1]	1	2	3	m[2]	1	2	3
12	5	43	11	3	23	84	12	5	7	9	10

Fig. 7.10

Si on traite le tableau ***m*** comme un tableau unidimensionnel chaque élément de ce tableau est un autre tableau unidimensionnel dont les éléments sont du type de base. Donc on peut voir que la valeur de ***m*** est l'adresse de la première ligne ***m[0]*** dont la valeur est l'adresse du premier élément du tableau bidimensionnel. Cettes relations sont mntrées ci-dessous.

```
m ≡ &m[0] ≡ &(&m[0][0])
m[0][0] ≡ *m[0] ≡ **m
```

Initialisation des tableaux bidimensionnels

Lorsque le tableau comporte des sous-tableaux, c'est-à-dire lorsque c'est un tableau à plusieurs indices, la liste des expressions d'initialisation peut comporter des sous-listes indiquées à leur tour avec des accolades, mais ce n'est pas une obligation. Le cas échéant, la règle de complétion par des zéros s'applique aussi aux sous-listes. S'il n'y a pas des sous-listes le tableau est initialisé comme un tableau uni dimensionel dont la même mémoire est allouée

Par exemple, les déclarations allouent et garnissent les deux tableaux de la figure 7.11.

```
int t1[4][4] = { { 1, 2, 3 }, { 4, 5, 6 }, { 7, 8, 9 } };
int t2[4][4] = { 1, 2, 3, 4, 5, 6, 7, 8, 9 };
```

t1

1	2	3	0
4	5	6	0
7	8	9	0
0	0	0	0

t2

1	2	3	4
5	6	7	8
9	0	0	0
0	0	0	0

Fig. 7.11

L'algorithme générique du traitement du tableau bidimensionnel

Ici on utilise deux fois l'algorithme générique pour traiter le tableau uni dimensionnel. D'abord on l'utilise pour traiter les lignes (les éléments du tableau) et puis chaque ligne est traitée par une boucle car elle est aussi un tableau. De tel façon l'algorithme représente deux boucles imbriquées.

```
for (i = 0; i<numlin; i++){
  //Traitement de ligne i
  for(j=0; j<numcol;j++){
    // Traitement de l'élément i,j
 }
}
```

Exemple :

Affichage d'une matrice :

```
for (i = 0; i<3; i++){
  for(j=0; j<4;j++){
    printf("%3d ",m[i][j]);
 }
 printf("\n");
}
```

Déclaration d'un type nouveau

La déclaration des variables complexes comme les tableaux multidimensionnels est un peu longue et difficile. Et si nous avons à déclarer plusieurs objets identiques on doit répéter tous leurs attributs. Par exemple, si on doit déclarer trois matrices de 5 lignes et 10 colonnes on doit écrire :

```
float mat1[5][10], mat2[5][10], mat3[5][10];
```

En plus on peut faire des fautes. Pour simplifier ce type de déclarations en C existe l'instruction ***typedef*** qui définie un type nouveau. Elle a la forme

```
typedef déclaration
```

Dans la déclaration le rôle du nom de la variable est joué par le nom du type qu'on veut définir.

Exemple :

```
typedef float MATRICE[5][10];
...
MATRICE mat1, mat2;
MATRICE mat3;
```

Paramètres de fonctions

Les raisonnements pour les tableaux unidimensionnels sont valides et pour les tableaux bidimensionnels mais avec une particularité – dans la déclaration du paramètre formel on doit définir le nombre de colonnes pour que on calculer correctement les adresses de débuts des lignes.

```
#define MAXCOL 10
#define MAXROW 20
typedef float MATRICE[ MAXROW][MAXCOL];
float floatsomme(  float  a[][MAXCOL],  int nl, int nc){
  int i,j;
  float s = 0;
  for (i=0; i<nl;i++)
       for (j=0; j<nc; j++)
             s+=a[i][j];
  return s;
}
```

L'instruction *goto*

Il y a des rares cas quand on doit sortir brusquement d'une boucle interne (imbriquée) hors de la boucle externe. Dans ce cas on ne peut utiliser l'instruction ***break*** car elle va sortir hors de la boucle interne mais dedans la boucle externe. Ici on utilise l'instruction ***goto*** qui a la syntaxe suivante :

```
goto label
      ...
label:instruction;
```

où *label* est un identificateur unique.

Exemple :

Trouver la somme de premiers L nombres positifs d'une matrice (Fig.7.12)

```
float floatsommeL(float a[][MAXCOL],
       int nl, int nc){
  int i,j,cont=0;
  float s = 0;
  for (i=0; i<nl;i++)
   for (j=0; j<nc; j++){
     if (a[i][j]>0) {
       s+=a[i][j];
       cont++;
     }
   if (cont == L) goto ex;
  }
  ex: return s;
}
```

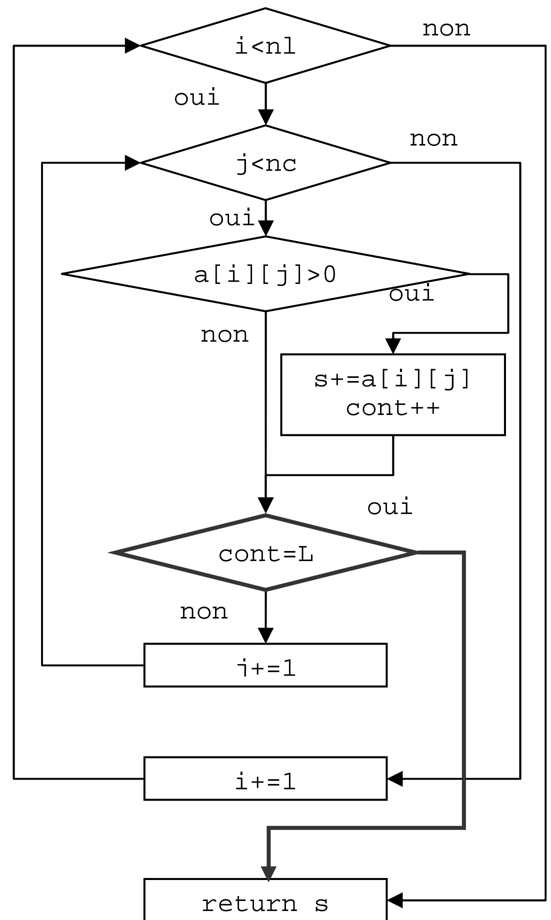


Fig. 7.12

Tableaux de pointeurs

C'est la deuxième manière de présenter un tableau bidimensionnel. Dans ce cas on déclare un tableau unidimensionnel de pointeurs vers objets de type de base. Puis à chaque pointeur est affectée l'adresse du premier élément de la ligne correspondante. Cette représentation n'exige pas que toutes les ligne sont mis à la même place dans la mémoire et d'autre coté d'avoir la même talle. Un exemple est montré au dessous (Fig.7.13) avec la disposition des objets dans la mémoire. On voit bien qu'il est nécessaire de stocker quelque part les tailles des lignes (le tableau *TAILLE*).

```
#include <stdio.h>
  int TAILLE[]={1,2,3};
  int ligne1[]={2};
  int ligne2[]={3,4};
  int ligne3[]={1,2,3};
void affichemat( int *a[], const int n);
void main (void) {
  int *mattriang[]={ligne1,ligne2,ligne3};
  affichemat(mattriang,3);
}
void affichemat( int *a[], const int n){
  int i,j;
  printf("Les  éléments du tableau\n");
  for (i=0; i<n; i++)  {
       for (j=0; j<TAILLE[i]; j++)
        printf("%3d, ",a[i][j]);
       printf ("\n");
  }
}
```

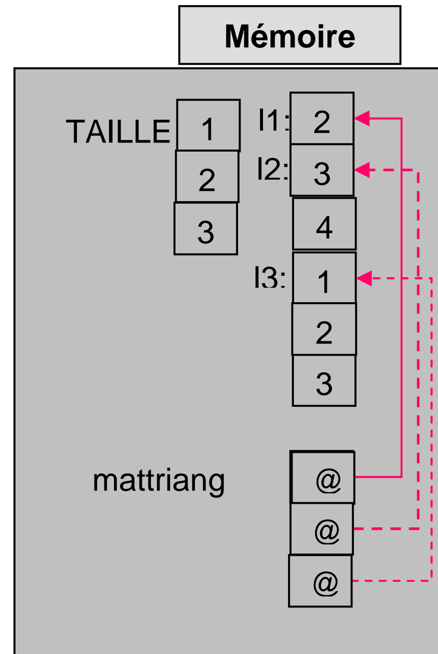


Fig. 7.13