Les tableaux

1 Déclaration/Initialisation d'un tableau

Tableau : agglomérat homogène (regroupement de variables de même type)
Objectif des tableaux : Gérer une collection de variables d'un même type.

Déclaration d'un tableau

```
type identificateur[nombre_composantes];
```

`type` : type de chaque composante (case) du tableau
`identificateur` : nom du tableau
`nombre_composantes` : valeur entière **constante** qui donne le nombre total de cases du tableau

Exemple 1:
`int tab[5];`
est une déclaration qui réserve une zone mémoire contiguë qui peut être représentée comme suit :

numéro de la case	1ère case	2e case	3e case	4e case	5e case
Identificateur : tab					
indice pour accéder à la case	0	1	2	3	4

Elle comporte 5 cases. Chaque case peut contenir une valeur entière de type int. Il ne faut pas confondre le numéro d'un élément du tableau et l'indice de la case qu'il occupe.
Le compilateur alloue la zone mémoire en contigu, mais n'initialise pas le contenu des cases.
Chaque case du tableau pourra contenir un entier.
Puisque toutes les cases sont stockées les unes à la suite des autres, l'adresse d'une case n°i est égale à l'adresse de la case précédente + 1.

Exemple 2 :
`char mot[9] ;`

numéro de la case	1ère cas	2e case	3e case	4e case	5e case	…	9e case
identificateur : mot							
indice pour accéder à la case	0	1	2	3	4	…	8

Chaque case peut contenir un caractère.

Généralement on associe une constante par `#define` pour définir la taille d'un tableau.
Exemple

```
#define NB_COMP 5 ;
…
int tab[NB_COMP];
```

définir une constante :
① dans un bloc const int nb_comp = 5;
② #define NB_COMP 5
au même niveau que #include ...

Accès à une composante par la valeur d'indice.

`identificateur[expression entière]`

l'expression entière correspond à la valeur d'indice

Ainsi si on reprend l'exemple 1

numéro de la case	1ère case	2e case	3e case	4e case	5e case
Identificateur : tab	tab[0]	tab[1]	tab[2]	tab[3]	tab[4]
indice pour accéder à la case	0	1	2	3	4

Si on reprend l'exemple précédent :

```
tab[0] = 25; /* rappel c'est la première case d'indice 0 */
tab[1] = 0;
tab[2] = 1;
tab[3] = -48;   tab[4] = -3227;
```

numéro de la case	1ère case	2e case	3e case	4e case	5e case
identificateur du tableau	25	0 ~~-2~~	1 ~~4~~	-48	-3227
indice pour accéder à la case	0	1	2	3	4

×

i = 2 tab[i] = 4; tab[i-1] = -2;

Attention : il faut toujours s'assurer que le l'expression entière entre crochets soit comprise entre `0` et `NB_COMP-1`

tab[5] provoque une erreur d'execution

Parcours d'un tableau

L'action favorite pour parcourir un tableau est une boucle for

```
for (i = 0 ; i< NB_COMP ; i++) {
          …tab[i]…
}
```

Initialisation du contenu des cases d'un tableau :

RAPPEL : comme toute variable une case d'un tableau doit être initialisée avant d'être utilisée.

Nous considérons 3 techniques pour initialiser le contenu des cases d'un tableau

* Initialisation à la définition : declaration

Exemple

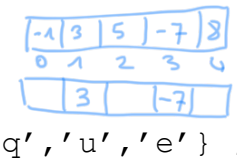
Ex 1 : `int tab[5] = {101, 102, -3, 44} ;`

Identificateur : tab	101	102	-3	44	
indice pour accéder à la case	0	1	2	3	4

La liste de valeurs d'initialisation est rangée dans l'ordre à partir de la case d'indice 0 (zéro).

Dans l'exemple 1 La case d'indice 4 n'a pas été initialisée

→ int tab[5] = {-1, 3, 5, -7, 8};
int tab[5] = { , 3, , -7};



Ex 2 :

```
#define TAILLE 9
char mot[TAILLE]  = {'v','e','r','o','n','i','q','u','e'} ;
```

identificateur : mot	'v'	'e'	'r'	'o'	'n'	'i'	'q'	'u'	'e'
indice pour accéder à la case	0	1	2	3	4	...			8

* Initialisation par lecture successive :

Exemple

```
int i ;
float tab[20];

printf("entrer 20 valeurs réelles ") ;
for (i=0 ; i < 20 ; i++)
          scanf(" %f ", &tab[i]);
```

Remarquons d'abord que l'on commence à l'indice 0 (zero), et que la dernière case que l'on va remplir est bien la case d'indice 19 = 20-1.

`tab[i]` est une variable de type entier ,donc `&tab[i]` est son adresse mémoire, ce dont à besoin le `scanf` pour ranger l'information lue au clavier puis formatée en entier (voir cours sur entrées/sorties).

* Initialisation par affectation directe (calcul):

Exemple 1

```
int i;
int U[20];
for (i = 0; i < 20 ; i ++)
     U[i] = 0;
```

Exemple 2

```
int i;
int U[20];
int a = 18;

U[0] = a;
U[1] = a + 1;

for (i = 2; i < 20 ; i ++)
     U[i] = U[i-1] + 2*U[i-2];
```

Nous avons initialisé les deux premières cases du tableau. Ensuite on a une boucle qui permet de travailler sur les indices variant de 2 à (N-1 = 19 = 20-1).

2. Occupation variable d'un tableau

constat :
* la taille d'un tableau doit être connue à la compilation
* la taille d'un tableau ne peut être modifiée au cours de l'exécution d'un programme

problème :
on ne connaît pas toujours le nombre d'éléments que l'on va ranger dans le tableau, mais on peut connaitre une borne max (pas plus de 100 par exemple)

Méthode :

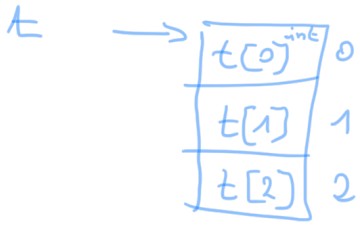
on définit une taille maximum (constante) MAX
on définit un tableau de taille MAX
on définit une variable qui indique le nombre d'éléments dans le tableau

Exemple :
```
#define MAX 1000 ;
int Tab[MAX] ;
int nbTab ;
```

ex :
```
printf ("donner le nbr de valeurs à mettre dans le
        tableau ( <=%d) SVP:", MAX);
scanf ("%d", &nbTab);
for (i=0; i<nbTab; i++) {
    printf ("entrer la %d eme valeur ", i+1);
    scanf ("%d", &Tab[i]);
}
```

la variable `nbTab` sera gére par le programmeur . elle signifie qu'on utilise uniquement les case d'indice compris entre `0` et `nbTab-1`

3. Relation pointeur-tableau



soit la déclaration de tableau suivante :
```
int t[3];
```

On a bien déclaré et réservé une zone mémoire contiguë (Cf. définition au début du cours), en fait :
l'identificateur `t` est l'identificateur d'un pointeur CONSTANT qui est de type 'pointeur vers int ', la valeur de cette constante est l'adresse du premier élément du tableau
`t` est de type `int *`
il contient la valeur `&t[0]`
il ne peut pas être modifié.

t <=> &t[0]
*t <=> t[0]

Ce qui a des conséquences logiques :
• On ne peut pas la modifier, c'est une constante

Exemple 8 :
```
int t[3];
int *pint;
int i = 5;

pint = &i;
t = pint; /* INTERDIT ! */
t = t + 1; /* INTERDIT */
```

arithmetique des pointeurs

int *p;

p+1 : adresse de l'entier qui suit l'entier d'adresse p


```
pint = t ; /* OK */
pint= pint+1 ;  /* OK */
pint = t+1 ; /* OK */
```

<u>* On ne peut pas utiliser directement l'opérateur d'affectation sur les tableaux</u>

Par exemple, si on veut faire la somme de 2 vecteurs v1 et v2, il faut donc ajouter les cases de v1 et v2 une à une.

```
int V1[5] = {1,2,3,4,5};
int V2[5];
V2 = V1;
```

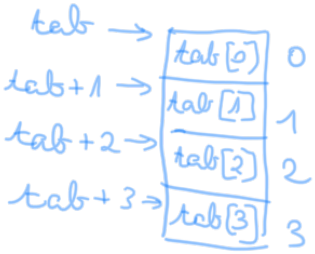
```
for (i=0; i<5; i++)
    V2[i] = V1[i];
```

Gràce à l'arithmétique des pointeurs (incrément des pointeurs), nous pouvons introduire une autre manière d'accéder à une case du tableau, par les pointeurs :

`t[i] = 0;` est équivalent à : `*(t+i) = 0;`

Nous avons donc 2 façons de réaliser l'accès à des cases d'un tableau :
`tab[i]` est équivalent à : `*(tab +i)`
`&tab[i]` est équivalent à : `tab +i`

La notation `tab[i]` est beaucoup plus facile à comprendre, à lire et à écrire.
c'est donc celle qu'il nous privilégierons dans ce cours.



exemple :

```
float truc[3] = {1.0, 2.0 ,3.0};
float * pf;

/* les deux lignes suivantes sont equivalentes */
pf = truc;
pf = &truc[0];

/* et les trois lignes suivantes sont aussi equivalentes */
*(pf+2) = -273.15;
*(truc+2) = -273.15;
truc[2] = -273.15;
```

4. Transmission d'un tableau en paramètres

Puisque l'identificateur du tableau déclaré est un pointeur constant, il faut utiliser un pointeur pour transmettre un paramètre de type 'tableau' dans une fonction.

Exemples déclaration d'une fonction qui attend un tableau en paramètre formel :

```
void somme_vect(int v[],int v1[],int v2[])
void somme_vect(int *v, int *v1, int *v2)
```

passage de parametre pour un tableau.

declaration fonction

.... fonction (type nom [])

appel de la fonction

type iden [taille];

.... fonction (iden)

quelque soit le rôle du tableau dans la fonction
- données
- données à modifier
- résultat

Toutes ces notations sont équivalentes. Nous n'avons pas besoin de connaître la taille d'un tableau. Nous avons juste besoin de savoir où il se trouve. Par contre la fonction aura besoin de la taille pour le parcourir : soit c'est une constante accessible par la fonction, soit elle est transmise en paramètre.

Exemple appel de la fonction avec les tableaux en paramètres.

```
somme_vect(a, b, c);
```
est équivalent à :
```
somme_vect( &a[0], &b[0], &c[0]);
```

Nous remarquons au niveau des deux appels de la fonction somme_vect que a dans le premier et que &a[0] au niveau du deuxième appel sont des adresses. Ce qui correspond au type de paramètre que nous avons défini dans la fonction.

Quelle que soit l'utilisation dans la fonction, on transmet généralement l'adresse du 1er élément du tableau.

Dans le corps de la fonction (le bloc), nous pouvons utiliser par exemple `v[i_local]` ce qui, nous l'avons vu, permet d'accéder au contenu de la case et de le modifier. Puisque le paramètre formel `v` contient une copie de `a` le pointeur constant qui est l'adresse du début de la zone mémoire allouée au sein de fonction principale main, nous modifions directement le tableau pointé par `a`.

en bref, une fonction ne peut pas retourner un tableau

```
#define TAILLE 10

void sommeVecteurs (int V[], int V1[], int V2[])
/* calcule dans V, V1+V2 */
{int i; /* parcours des tableaux */
 for (i=0; i<TAILLE; i++)
     V[i] = V1[i] + V2[i];
 return;
}

void saisieVecteur (int V[]) /* la fonction va remplir le tableau */
{ int i;
  for (i=0; i<TAILLE; i++) {
      printf ("donner la composante n° %d ", i+1);
      scanf ("%d", &V[i]);
```

```
    }
  return;
}
void afficheVecteur (int V[])   /* on suppose que V contient
{ int i;                          des valeurs */
  for (i=0; i<TAILLE; i++)
     printf("\t %d", V[i]);
}

 void main()
{ int T1[TAILLE], T2[TAILLE], T3[TAILLE];

  saisieVecteur (T1);
 saisieVecteur (T2);
 sommeVecteurs (T3, T1, T2);
  afficheVecteur (T3);
 return 0;
}
```