Algorithmique & Langage C
IUT GEII S1

Les Tableaux

Notes de cours

(quatrième partie)

4

cours_algo_lgc4.11.odp


Licence


•Paternité - Pas d'Utilisation Commerciale -
• Partage des Conditions Initiales à l'Identique 2.0 France

•Vous êtes libres :
• * de reproduire, distribuer et communiquer cette création au public
• * de modifier cette création, selon les conditions suivantes :
•
•**Paternité**. Vous devez citer le nom de l'auteur original.
•
•**Pas d'Utilisation Commerciale**.
•Vous n'avez pas le droit d'utiliser cette création à des fins commerciales.
•
•**Partage des Conditions Initiales à l'Identique**.
•Si vous modifiez, transformez ou adaptez cette création,
•vous n'avez le droit de distribuer la création qui en résulte
•que sous un contrat identiqueà celui-ci.
• * A chaque réutilisation ou distribution, vous devez faire apparaître clairement aux autres les
• conditions contractuelles de mise à disposition de cette création.
•* Chacune de ces conditions peut être levée si vous obtenez l'autorisation du titulaire des droits
•Ce qui précède n'affecte en rien vos droits en tant qu'utilisateur (exceptions au droit d'auteur :
•copies réservées à l'usage privé du copiste, courtes citations, parodie...)
•voir le contrat complet sous : http://fr.creativecommons.org/contrats.htm


Plan du cours

- Définition
- Déclarations
- Dangers des tableaux
- Algorithmes à connaître
- Tableaux en paramètres de fonctions


Définition

Jusqu'à présent les variables ne pouvaient contenir qu'une seule valeur (à un moment donnée) :

C'était des variables de type *scalaire*.
(int , float, double , ...)

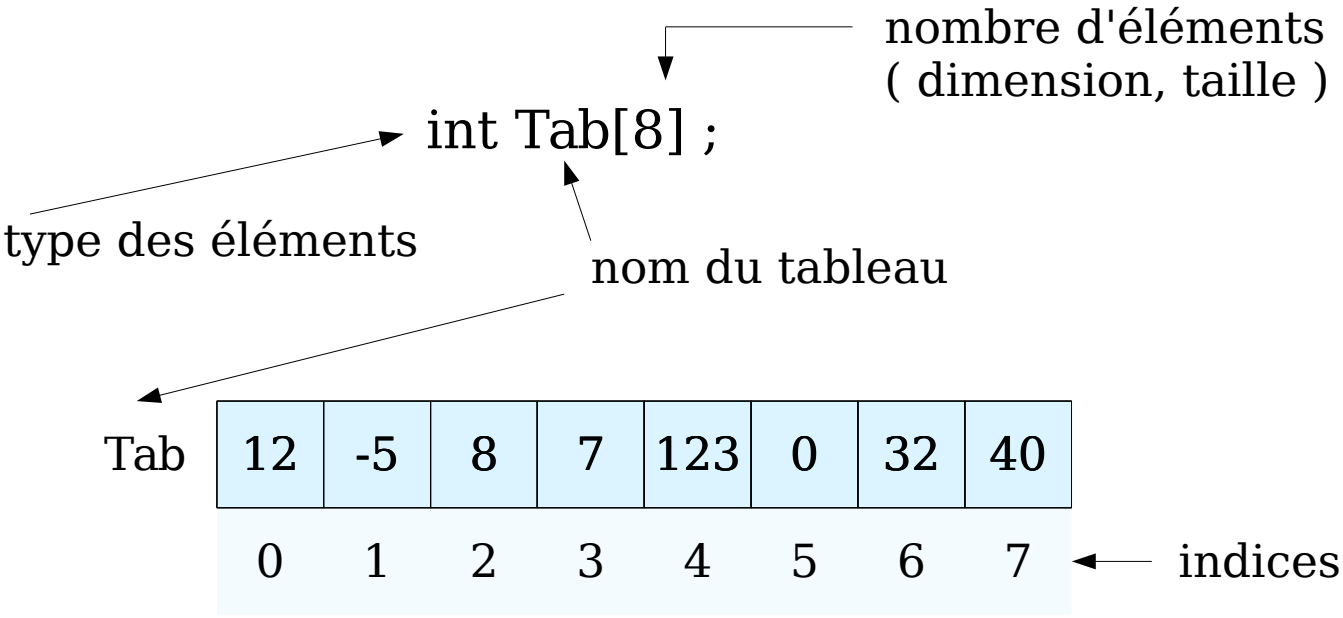
Un tableau est une suite ordonnée de valeurs du même type.

Ainsi une variable tableau est une variable qui pourra contenir plusieurs valeurs.

Autre nom :

En anglais :


Déclaration



ici, l'élément d'indice vaut il est noté **Tab[ ]**

Déclaration

<type des éléments > <nom du tableau> **[** <nombre d'éléments> **]** ;

Exemples :

```
double Vect1 [ 3 ] ;

float notes_de_math [ 80 ] ;

int Ta [ 24 ] ;
```

Comme pour les variables scalaires, les éléments d'un tableau déclarés dans un bloc ont des valeurs quelconques.

Déclaration et initialisation

Il est possible de donner des valeurs aux éléments lors de la déclaration d'un tableau :

Tous les éléments sont initialisés :

```
int Tab1[5] = { 2 , 3 , -3 , 6 , 7 } ;
```

Les deux derniers sont mis à 0 :

```
int Tab2[5] = { 2 , 3 , -3 } ;
```

2	3	-3	0	0

La taille est déduite de la liste d'éléments

```
int Tab3[] =  { 2 , 3 , -3 } ;     // exactement 3 éléments
```

Erreur trop d'éléments : `int Tab4[3] = { 2 , 3 , -3, 8 } ;`

Déclaration

La taille du tableau est une valeur fixe qui ne peut plus changer après la déclaration.

On ne peut pas rajouter ou enlever des éléments.

La taille est une valeur de type entier.

C89 : **La taille est une constante entière.**

C>99 : La taille peut être contenue dans une variable de type entier.
Mais elle reste fixe après la déclaration !

Accès aux éléments

L'instruction **int Tab[4] ;** déclare un tableau de **4 éléments numérotés de 0 à 3**.

Attention après la déclaration la valeur entre [ ] représente le numéro de l'élément (indice).

Les éléments sont numérotés de **0** à **taille – 1** .

```
Tab[0] = 3 ;
Tab[1] = 4 ;
Tab[2] = 2 ;
Tab[3] = 8 ;

   printf("%d",Tab[0] ) ;
```


Accès aux éléments

La valeur entre [] peut être n'importe quelle expression entière :

```
 int i = 0 ;
 float Vect[5] ;

 for ( i = 0 ; i < 5 ; i = i + 1 )
 {
     Vect[ i ] = 12.4 ;
 }
```

Comme le tableau Vect est de type float, Vect[i] se comporte comme une variable de type float. On peut donc l'utiliser comme n'importe qu'elle variable de type float :

```
printf("%f",Vect[i+3] );
if ( ( Vect[i] +  8 ) >= 16 ) { .....
```


Attention : Il n'y a pas de contrôle de débordement !

```
int Tab[4] ;          !!!
for ( i=0 ; i < 6 ; i = i + 1 ) {
   Tab[i] = 12 ;   }
```

Tab	12	12	12	12	12	12	
	0	1	2	3	4	5	

zone mémoire réservée à Tab

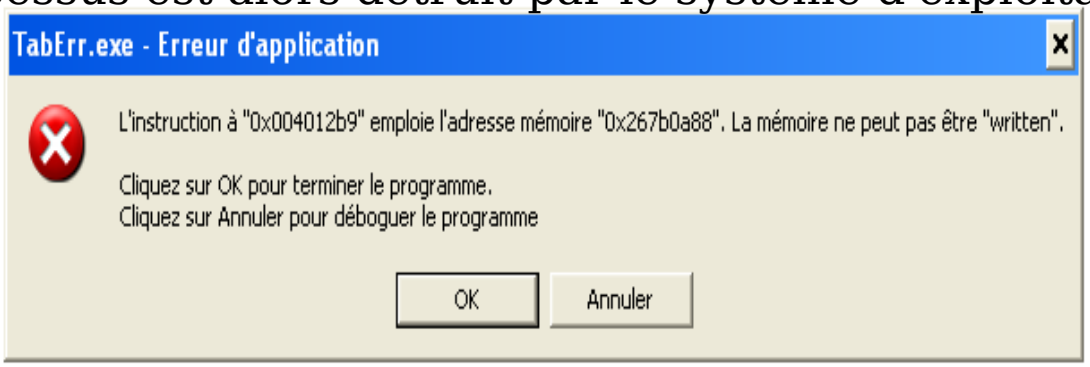
zone mémoire qui est peut être utilisée par d'autres variables ! Elles seront alors écrasées !

Attention : Il n'y a pas de contrôle de débordement !

Si on déborde du tableau il n' y a jamais d'erreur de compilation.
Par contre il peut y avoir des erreurs lors de l'exécution.

Si on déborde de beaucoup, on risque d'accéder à une zone mémoire inexistante ou réservée à un autre processus.

Le processus est alors détruit par le système d'exploitation.



Avec linux on a le message : segmentation fault

Attention : Il n'y a pas de contrôle de débordement !

Si on déborde de peu (quelques éléments), on peut écrire dans une zone mémoire qui est allouée à d'autres variables du programme. Dans ce cas le comportement du programme peut sembler incohérent.
C'est un bug sournois qui peut parfois passer inaperçu pendant longtemps.

```
int k = 111 ;
int  Tab[10];
int    i ;
for ( i=0 ; i<16 ; i=i+1 )
 {
     Tab[i] = 222 ;
 }
 printf("k=%d\n",k) ;
```

Compilé avec mingw (gcc pour windows), ce programme affiche 222 !


Mémoire avant et après le débordement

i	3
Tab[0]	222
Tab[1]	222
Tab[2]	222
Tab[3]	222
Tab[4]	??
Tab[5]	??
Tab[6]	??
Tab[7]	??
Tab[8]	??
Tab[9]	??
k	111

i	16
Tab[0]	222
Tab[1]	222
Tab[2]	222
Tab[3]	222
Tab[4]	222
Tab[5]	222
Tab[6]	222
Tab[7]	222
Tab[8]	222
Tab[9]	222
	222
	222
k	**222**
	222

k a été écrasé!


Algorithmes à connaître

- Saisir la valeurs des éléments au clavier.
- Afficher la valeurs des éléments.
- Calculer la somme des éléments.
- Recopier les éléments d'un tableau dans un autre.
- Compter le nombre d'éléments qui vérifient une propriété donnée.
- Trouver la plus grande valeur d'un tableau (maximum).

Saisir la valeurs des éléments au clavier

```
float tab[5];
int i=0;

while( i < 5 )
{
printf("Entrez la valeur numero %d : ",i);
scanf("%f",&tab[i]);
i = i + 1 ;
}
```

Afficher la valeurs des éléments

```
float tab[5] ;
i=0;
..... ;
while ( i < 5 )
{
printf("l'element %d vaut %f",i,tab[i]);
i = i + 1 ;
}
```

Calculer la somme des éléments

```
float somme = 0 ;
float tab[5] ;
i=0;
..... ;
while ( i < 5 )
{
somme = somme + tab[i] ;
i = i + 1 ;
}
```

Ne pas oublier l'initialisation à 0 !

Recopier les éléments d'un tableau dans un autre

```
// recopie des éléments de tab1 dans tab2
float tab1[5] ;
float tab2[5] ;
i=0;
..... ;
while ( i < 5 )
{
tab2[i] = tab1[i] ;
i = i + 1 ;
}
```

taille de tab2 >= taille de tab1

Exercice : Recopier "à l'envers"
(premier -> dernier, deuxième -> avant dernier, ...,...,...,dernier->premier)


Compter le nombre d'éléments qui vérifient une propriété donnée

```
//Compter le nombre d'éléments positifs

float tab[5] ;
int nb_pos = 0 ;
i=0;
```

Ne pas oublier l'initialisation à 0 !

```
while ( i < 5 )
{
  if ( tab[i] > 0 ) {
            nb_pos = nb_pos + 1 ;
        }
i = i + 1 ;
}
```


Trouver la plus grande valeur d'un tableau (maximum)

```
float tab[5] ;
float max ;
saisie des éléments...;
int i = 1;
max = tab[0] ;
while ( i < 5 )
{
  if ( tab[i] > max ) {
            max = tab[i] ;
        }
i = i + 1 ;
}
```

max est par définition une valeur du tableau !

Attention : Ne pas initialiser max à 0. Si tous les éléments sont négatifs, on va trouver max égal à 0 !

Fonctions ayant des tableaux en paramètres

Fonction retournant la moyenne des valeurs d'un tableau de 10 éléments de type float :

Prototype :

float moyenne_tab (float []) ;

ou

float moyenne_tab (float *) ;

Attention il n'y a pas d'indication ni de contrôle de la taille du tableau. En fait c'est seulement l'adresse du premier élément qui est passée à la fonction.

Fonctions ayant des tableaux en paramètres

Définition de moyenne_tab :

```
float moyenne_tab ( float tab [ ]  ) {
   int i  ;
   float s = 0 ;
   for ( i = 0 ; i < 10 ; i = i + 1 ) {
     s = s + tab[i] ; }
   s = s / 10 ;
   return s ;
}
```

Appel :

```
float t[10] ;
float moy ;
moy =   moyenne_tab( t  ) ;
```


Fonctions ayant des tableaux en paramètres

La fonction précédente ne peut traiter que des tableaux de 10 éléments. En passant la taille en paramètre, on peut rendre la fonction plus générale :

```
float moyenne_tab ( float * tab  , int taille  )  ;

float moyenne_tab ( float  * tab  , int taille  ) {
   int i  ;
   float s = 0 ;
   for ( i = 0 ; i < taille ; i = i + 1 ) {
     s = s + tab[i] ; }
   s = s / taille ;
   return s ;
}
```

Appel :

```
   moy =  moyenne_tab ( t , 10  ) ;
```


La fonction peut modifier les éléments d'un tableau

Comme on passe l'adresse du premier élément, une fonction peut agir directement sur les éléments d'un tableau.
Fonction qui remplace les valeurs supérieures à max par max :

Prototype : **void EcreteTab (int * tab, int max ,int taille) ;**

Définition :

```
void EcreteTab ( int * tab ,  int max ,int taille ) {
        int i ;
        for ( i = 0 ; i < taille ; i++ ) {
                if ( tab[i] > max ) {
                        tab[i] = max ;  }
        }
}
```

Appel :

```
int t[]={5,6,7,34,2,3,45,6,67,99};
EcreteTab(t,45,10);
```


Prototypes : const , noms de paramètres

On peut ajouter le mot clé **const** devant le type d'un paramètre tableau pour indiquer que la fonction ne modifie pas la valeur des éléments du tableau.

float moyenne_tab (const int * , int) ;

On peut nommer les paramètres dans le prototype d'une fonction pour se rappeler leur signification :

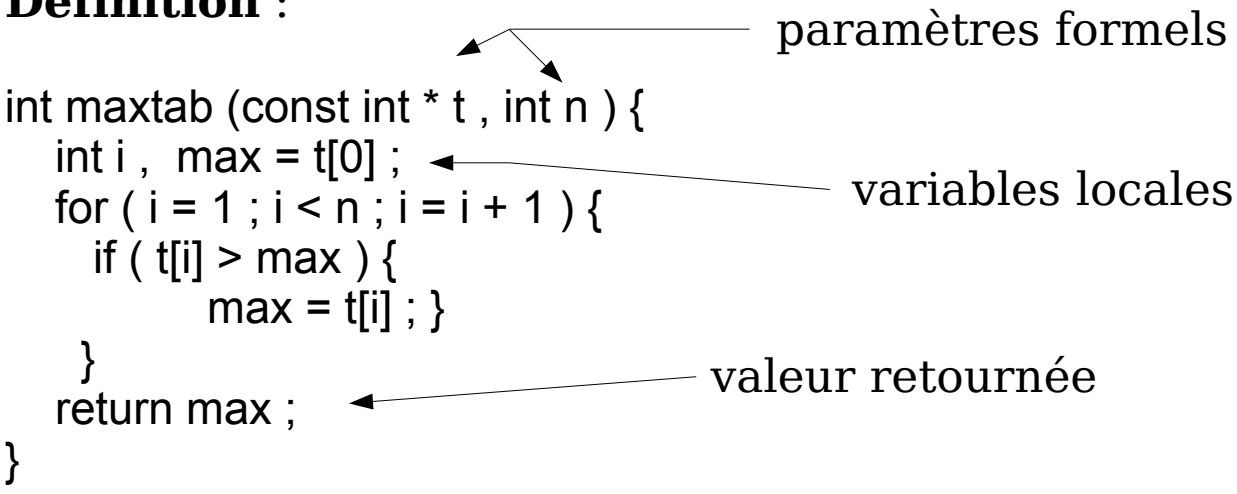
float moyenne_tab (const int * tableau , int taille) ;

(les noms sont alors vus comme des commentaires par le compilateur)

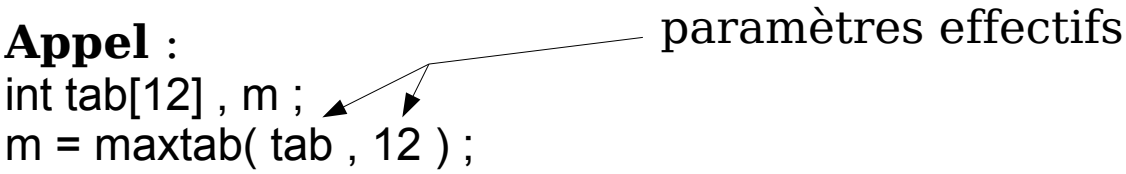

Vocabulaire

Prototype : int maxtab (const int * t , int n) ;

Définition :



Appel :



Exercices

- Comparer deux tableaux d'entiers
- Rechercher si une valeur existe parmi les éléments d'un tableau
- Remplir un tableau de 1000 entiers avec les 1000 premiers nombres premiers. Ensuite utiliser les éléments de ce tableau pour déterminer efficacement la primalité d'un nombre.
- Rechercher la valeur immédiatement inférieure au maximum d'un tableau.