- Introduction à l'algorithmique et au langage C -

Introduction aux tableaux

TD5

1re année ESIEA - Semestre 1

L. Beaudoin & R. Erra & A. Gademer & L. Avanthey

2017 - 2018

Avant propos

Il est temps maintenant d'introduire une nouvelle structure de donnée : les tableaux ! Nous allons voir comment les déclarer et les initialiser. Puis nous commencerons à aborder quelques petits algorithmes bien utiles qui leurs sont associés. Enfin nous verrons l'utilisation combinée des fonctions et des tableaux.

1 Objectif

Nous avons vu dans les TD/TP précédents comment déclarer des variables pour stocker des informations en mémoire. Mais que se passe-t-il si j'ai un grand nombre d'informations à stocker : l'ensemble des notes au module INF1031 pour la promotion (une centaine d'entiers), la température journalière à Honolulu les 10 dernières années (plus de 3 000 flottants), les votes électroniques de la présidentielle (20 ou 30 millions d'entiers) ? Il est impensable de déclarer indépendamment 1 million de variables (`int var1; int var2; ...`). C'est pourquoi nous allons faire appel aux **tableaux**. Les tableaux sont des structures de données permettant de déclarer d'un bloc une grande quantité de variables **d'un même type**. Nous pouvons alors utiliser toutes ces cases mémoire pour stocker nos informations.

Ne mélangez pas torchons et serviettes

Les tableaux regroupent une grande quantité de variables d'un **même** type. Cela signifie évidement qu'il est **impossible** de déclarer un tableau qui contient à la fois des entiers et des flottants ou des caractères.

2 Déclaration

La déclaration d'un tableau est similaire à celle d'une variable avec le **type** suivi du **nom** du tableau, à laquelle nous rajoutons **le nombre de cases** du tableau **entre crochets**.

```
int myVariable; // Reserves 1 variable of type int
int myArray[10]; // Reserves 10 variables of type int
```

QUESTION 1


Écrivez la déclaration d'un tableau de quinze flottants, puis celle d'un tableau de 80 caractères.

Nous accédons aux variables réservées à partir du nom du tableau grâce à **l'indice** de la case (c'est-à-dire son numéro) que nous précisons entre crochets.

```
myArray[3] = 12; // Set the value of the element at index 3 to 12
printf("%d", myArray[4]); // Show the value of the element at index 4
```

Attention aux indices !
En C, les indices des tableaux commencent **toujours** à **zéro** (0). Cela implique que l'indice de la **dernière case** d'un tableau de taille `N` est `N-1`. (0, 1, 2, 3 ⇒ 4 cases.)

```
myArray[0] = 5; // OK, first element has index 0
myArray[10] = 7; // KO, last element (of an size 10 array) has index 9
```

Accès interdit, vous n'avez pas réservé !
Avec cette dernière instruction, vous essayez d'atteindre un emplacement mémoire qui n'existe pas car vous ne l'avez pas réservé ! Vous avez demandé 10 cases (de 0 à 9) et non 11... Faites donc très attention à ne jamais sortir du tableau avec vos indices !

3 Initialisation

Si nous pouvons facilement déclarer un tableau de grande taille avec une seule instruction, initialiser les éléments de ce tableau demande en revanche une plus grande astuce.

Solution n°1 : Nous affectons les valeurs à chaque case, une par une.

```
int myFirstArray[4];
myFirstArray[0] = 1; // We set the first element to 1
myFirstArray[1] = 2; // We set the second element to 2
myFirstArray[2] = 3; // We set the third element to 3
myFirstArray[3] = 4; // We set the fourth element to 4
```

Solution n°2 : Nous initialisons toutes les cases du tableaux avec la notation « accolade » (uniquement valable en combinaison avec la déclaration) :

```
int mySecondArray[4] = {1, 2, 3, 4}; // The first element is set to 1, the second to 2, the
    third to 3 and the fourth to 4
```

QUESTION 2


Écrivez la déclaration :

a) d'un tableau de 10 flottants initialisés de 10.0 à 1.0,

b) d'un tableau de 6 caractères initialisés avec les voyelles en minuscules.

4 Algorithmes sur les tableaux

Nous trouverons généralement trois grandes catégories d'algorithmes associés directement aux tableaux : les algorithmes de parcours, de recherche et de tri.

4.1 Parcours

Il est très utile de pouvoir parcourir tous les éléments d'un tableau. Et pour cela nous allons utiliser l'une des structures de contrôles que nous avons apprises : les boucles ! Voyons voir : nous prenons le premier élément, nous faisons quelque chose avec (nous le regardons, nous lui donnons une valeur, etc.), puis nous prenons l'élément suivant, nous faisons les mêmes choses avec, puis l'élément suivant, ... jusqu'au dernier élément. C'est tout à fait cela, nous avons bel et bien une boucle !

Itérative (`for`) ou événementielle (`while`) ?
Alors creusez vos méninges, à quel type de boucle avons-nous affaire ?

Dans le mille, une boucle itérative car nous connaissons à l'avance le nombre d'éléments du tableau. Pour tous ceux qui ont pensé « événementielle », vous avez gagné le droit de révisez la formule magique : « quand nous connaissons le nombre de tours de boucle, c'est itératif, sinon c'est événementiel » ! Donc nous utilisons une boucle `for`, avec un compteur qui va de 0 (le premier indice) à `taille-1`.

Ce type de parcours nous sera très utile pour afficher un tableau mais aussi pour l'initialiser. Les méthodes que nous avions vues un peu plus haut sont très bien mais incomplètes. Que faisons-nous si nous voulons remplir un tableau de 1 000 entiers avec les chiffres de 1 à 1 000 ? Aucune des deux solutions n'est vraiment satisfaisante et entrainerait de sacrées crampes dans les doigts. Mais la boucle nous résout ce problème : en parcourant tous les éléments nous pouvons leur affecter une valeur donnée (celle d'un compteur, une valeur aléatoire, le résultat d'un calcul, etc.) !

QUESTION 3

 Déclarez un tableau de 10 entiers puis écrivez le code permettant de parcourir ce tableau et d'initialiser les éléments avec les chiffres de 0 à 9.

QUESTION 4

 Même question, mais nous voulons cette fois-ci initialiser les éléments avec les chiffres de 1 à 10.

QUESTION 5

 Même question, mais nous voulons cette fois-ci initialiser les éléments avec les chiffres de 10 à 1.

QUESTION 6


Écrivez le code qui permet d'afficher le contenu du tableau.

4.2 Recherche

L'avantage majeur des tableaux est de pouvoir accéder directement à une information (sans passer par les précédentes) **à condition de connaître l'indice de la case** où elle se trouve.

Mais il arrive que nous ne connaissions pas dans quelle case est rangée l'information qui nous intéresse. Il nous faut alors utiliser des algorithmes de **recherche**. En résumé, un algorithme de recherche est formé d'un **parcours** et d'un **test** qui va déterminer si l'élément actuellement parcouru est celui que nous cherchons. Selon les circonstances, la boucle pourra alors être brisée (`break`) pour éviter de parcourir inutilement le reste du tableau.

Pour les questions suivantes, nous considérerons les variables ci-après :

```
int myExampleArray[10] = {9, 6, 10, 7, 9, 3, 5, 12, -4, 8};
int seekedElementIndex;
int seekedElementValue;
```

QUESTION 7


Écrivez le code qui permet de parcourir ce tableau et d'affecter à la variable `seekedElementIndex` l'indice de la première case contenant le nombre 3. Si ce nombre n'est pas présent dans le tableau, vous affecterez -1 à la variable.

QUESTION 8


Écrivez le code qui permet de parcourir ce tableau et d'affecter à la variable `seekedElementIndex` l'indice de la première case négative, ou -1 si aucun nombre de ce type n'est présent dans le tableau.

QUESTION 9


Écrivez le code qui permet de parcourir ce tableau et d'affecter à la variable `seekedElementValue` la valeur de l'élément le plus grand du tableau.
Indice : conservez la valeur du maximum temporaire (la plus grande valeur rencontrée jusqu'ici lors du parcours).

Attention

N'initialisez pas le maximum temporaire avec une valeur arbitraire, vous ne pouvez pas préjuger du contenu du tableau. Le maximum pourrait être -57.

QUESTION 10


Écrivez le code qui permet de parcourir ce tableau et d'affecter à la variable `seekedElementIndex` **l'indice** de l'élément le plus grand du tableau.
Indice : conservez la valeur **et l'indice** du maximum temporaire (la plus grande value rencontrée jusqu'ici lors du parcours).

4.3 Tri

Pour finir, nous allons voir la dernière catégorie d'algorithme de manipulation de tableau : les algorithmes de tri. En effet, comme nous l'avons vu dans les exemples précédents, les éléments d'un tableau ne sont pas rangés automatiquement. Nous pouvons avoir besoin qu'ils soient ordonnés du plus petit au plus grand (tri croissant) ou inversement (tri décroissant).

Il existe de nombreux algorithmes de tri qui varient selon leur complexité et leur efficacité. Pour le moment, nous nous contenterons d'un algorithme naïf : **le tri à bulles**. Au second semestre, quand nous aurons commencé l'analyse d'algorithmes, nous en découvrirons d'autres.

Le principe du tri à bulles est très simple : un tableau est trié de manière croissante si chacune des cases est plus petite que sa voisine de droite. Nous allons donc parcourir notre tableau et comparer chaque case à sa voisine de droite. Si elle est plus grande, les éléments sont échangés (rappelez-vous, nous avons vu l'algorithme d'échange de variables dans le TD « *Hello World !* »).

Arrivé au bout du parcours, deux possibilités s'offrent à nous : soit nous n'avons échangé aucun élément et le tableau était trié (nous nous arrêtons), soit nous avons échangé au moins un couple et il nous faut recommencer notre test du début (un seul passage ne garanti pas le tri, essayez sur une feuille de papier).

Deux pour le prix d'une

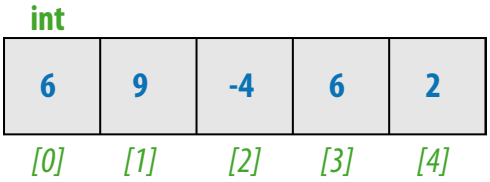
Nous allons donc avoir une boucle événementielle (avons-nous fait au moins un échange ?) autour de notre boucle de parcours qui est itérative ! Pas de panique, c'est ce que nous appelons une **double boucle**.

Redonnons l'algorithme du tri à bulle.

répéter
  Affecter vrai à "estTrié"
  pour indice de 0 à n-2 inclu **faire**
    si case [indice] > case [indice + 1] **alors**
      Échanger case [indice] et case [indice + 1]
      Affecter faux à "estTrié"
    fin
  fin
tant que "estTrié" est faux

QUESTION 11

Appliquez l'algorithme du tri à bulle sur le tableau suivant.



Parcours n°	Étape	[0]	[1]	[2]	[3]	[4]	estTrié ?
1	`isSorted=1`	6	9	-4	6	2	1
1	[0] > [1]	6	9	-4	6	2	1
1	[1] > [2]	6	-4	9	6	2	0

(Suite page suivante)

Parcours n°	Étape	[0]	[1]	[2]	[3]	[4]	estTrié ?
1	[2] > [3]						
1	[3] > [4]						
2	`isSorted=1`						
2	[0] > [1]						
2	[1] > [2]						
2	[2] > [3]						
2	[3] > [4]						
3	`isSorted=1`						
3	[0] > [1]						
3	[1] > [2]						
3	[2] > [3]						
3	[3] > [4]						
4	`isSorted=1`						
4	[0] > [1]						
4	[1] > [2]						
4	[2] > [3]						
4	[3] > [4]						

5 Passage de tableau en paramètre d'une fonction

Dans ce TD, nous avons vu comment déclarer et utiliser les tableaux. Nous verrons la représentation exacte des tableaux en mémoire lors d'un prochain TP, mais nous pouvons déjà voir **comment passer un tableau en paramètre d'une fonction**.

Tout d'abord, afin de connaître le nombre de cases du tableau, nous passons un premier paramètre qui correspond à la **taille du tableau**, puis nous passons le tableau lui-même.

En paramètre, un tableau se déclare avec son type, son nom et des **crochets vides** qui indiquent qu'il s'agit d'un tableau et non d'une simple variable.

```
void myFunction(int size, int myArray[])
```

En argument, nous passons un tableau avec **son nom uniquement** (sans les crochets).

```
int myArray[10] = {0};
myFunction(10, myArray);
```

L'utilisation au sein de la fonction correspond à l'utilisation classique des tableaux.

```
void showArray(int size, int myArray[]) {

   int i;

   for (i = 0; i < size; i++) {
      printf("%d " , myArray[i]);
   }

   printf("\n");
```

```
}
```

Attention

Contrairement aux variables classiques, les éléments d'un tableau passé en argument d'une fonction peuvent être modifiés par celle-ci !

```
void initArray(int size, int myArray[]) {

   int i;

   for (i = 0; i < size; i++) {
      myArray[i] = i*i;  // 0, 1, 4, 9, 16, ...
   }

}

int main() {

   int iArray[10] = {0}; // All elements are set to 0
   initArray(10, iArray); // After this, all elements will be set to 0, 1, 4, 9, 16, ...
   return 0;

}
```

QUESTION 12

Donnez le prototype de la fonction `swap` qui prend un tableau d'entiers et deux indices en paramètre et qui ne renvoie rien car elle ne fait qu'échanger les deux éléments correspondant aux indices donnés dans le tableau (souvenez-vous que le tableau, contrairement aux variables, restera modifié à la sortie de la fonction).

QUESTION 13

Écrivez en C l'algorithme du tri à bulle en considérant que vous avez accès à la fonction d'échange `swap` dont vous avez donné le prototype ci-dessus.

(suite)

QUESTION 14


À votre avis, pourquoi cet algorithme est appelé tri à bulles ?

6 Ils sont fous ces romains

La guerre des Gaules fait rage. L'issue est incertaine. Pour mettre toutes les chances de son coté, César décide de transmettre les messages stratégiques de manière chiffrée pour limiter les fuites d'information vers l'ennemi. Le principe est simple, nous remplaçons l'alphabet classique par un alphabet décalé de 3 lettres comme illustré au tableau suivant. Ainsi, le mot « *bac* » devient, une fois chiffré, « *edf* ». Lumineux, non ?

Alphabet normal	a	b	c	d	e	f	...	v	w	x	y	z
Alphabet codé	d	e	f	g	h	i	...	y	z	a	b	c

D'un point de vue plus formel, en supposant que la lettre « a » corresponde à la valeur 0 et « z » à la valeur 26, alors la fonction de chiffrement de César revient à :

$$Cesar(x) = (x + 3) \text{ modulo } 26 \tag{1}$$

Pour décrypter le message, on prend la fonction inverse :

$$Cesar^{-1}(x) = (x - 3 + 26) \text{ modulo } 26 \tag{2}$$

Modulo

Notez l'ajout de la valeur 26 pour compenser le décalage négatif. En effet, l'opérateur % (reste de la division euclidienne) n'aime pas les nombres négatif. En C, $-3\%26 = -3$, contrairement à la notion de congruence mathématique. D'un point de vue mathématique, l'opération est transparente puisque $(a + b) \bmod b \Leftrightarrow a \bmod b + b \bmod b \Leftrightarrow a \bmod b$

6.1 Notre pierre de Rosette : la table ascii

Comme nous l'avons déjà indiqué, un ordinateur ne sait manipuler que des nombres. C'est pourquoi en machine, chaque caractère est associé à un nombre unique par une table de codage. La table la plus répandue est la table ASCII [1] qui associe les caractères courants (non accentués) aux nombres de 0 à 127.

Nous pouvons visualiser l'ensemble de ces correspondances caractère-nombre de la table ASCII en entrant la commande `man ascii` dans la console.

Voici un extrait de la table ASCII :

'+'	...	'0'	'1'	...	'9'	...	'A'	'B'	...	'Z'	...	'a'	'b'	...	'z'
43		48	49		57		65	66		90		97	98		122

Nous pouvons remarquer :
- que la disposition des caractères est arbitraire (en particulier pour les symboles),
- que les chiffres et les lettres restent ordonnés et contigus par bloc.

Ce qui signifie que :
- `'0' + 9 = 48 + 9 = 57 = '9'`
- `'A' + 25 = 65 + 25 = 90 = 'Z'`
- `'m' - 3 = 109 - 3 = 106 = 'j'`

QUESTION 15

Décryptez à la main le message suivant : `Tbii\alkb`, qui a été crypté par la méthode de César.

Message codé	Z	h	o	o	b	g	r	q	h
Message décodé									

-3

EXERCICE 1

Créez un fichier `caesar.c` et écrivez-y le code du `Hello World!`.

EXERCICE 2

Ajoutez la fonction `getIntegerGreaterThan` que vous avez déjà codée plusieurs fois et qui réitère la demande `"Please enter a value greater than XX:\n"` (ou XX correspond au seuil `threshold` passé en paramètre) tant que l'entier récupéré au clavier est inférieur ou égal au seuil (en cas de saisie incorrecte, il faut afficher `"Input error\n"` puis quitter le programme).

EXERCICE 3

Appelez cette fonction dans la fonction `main` et déclarez un tableau de taille correspondante à l'entier récupéré. Notez que ce sera le seul cas de figure où vous ferez une déclaration dans la zone d'instruction (ici, c'est obligatoire car nous devons connaître la taille pour pouvoir déclarer le tableau).

EXERCICE 4

Ajoutez la fonction `getACharacter` qui demande `"Please enter a character\n"` et permet de récupérer un caractère (différent du retour à la ligne `'\n'`) au clavier et le renvoie. Si le caractère récupéré est un retour à la ligne, renouvelez la demande mais sans ré-afficher le message (en cas de saisie incorrecte, il faut afficher `"Input error\n"` puis quitter le programme).

1. American Standard Code for Information Interchange. Nous noterons l'existence de nombreuses autres tables qui permettent d'ajouter les caractères accentués comme la table Latin1 ou encore les caractères cyrilliques, chinois, etc., comme la table Unicode.

Récupération de caractères et retour à la ligne

Le retour à la ligne ou l'espace servent habituellement à séparer les nombres que nous désirons récupérer avec le `"%d"` mais ici, nous cherchons à récupérer des caractères, du coup `'\n'` et `' '` sont des valeurs acceptables ! Du coup, à chaque fois que vous tapez un caractère suivi d'un retour à la ligne, il considère que vous avez saisie 2 caractères. C'est pourquoi nous avons choisi la consigne précédente.

EXERCICE 5

Ajoutez la fonction `loadCharArray` qui prend en paramètre une taille `size` et un tableau de caractères associé `charArray`) et qui rempli les cases du tableau avec des caractères récupérés au clavier (utilisez la fonction `getACharacter`).

EXERCICE 6

Appelez cette fonction dans la fonction `main` pour initialiser votre tableau avec des caractères.

EXERCICE 7

Ajoutez la fonction `caesarCypher` qui prend en paramètre un caractère `character` et qui retourne le caractère chiffré selon l'algorithme de César (soit avec le décalage 3).

Nombre de symboles

Attention ! Si nous utilisions la congruence modulo 26 pour les 26 lettres de l'alphabet, la table ASCII, quant-à-elle, comporte 128 symboles.

EXERCICE 8

Ajoutez la fonction `caesarCypherArray` qui prend en paramètre une taille `size` et un tableau de caractère `charArray` et qui modifie chacune des valeurs du tableau en la remplaçant par le caractère chiffré selon l'algorithme de César (soit avec le décalage 3).

EXERCICE 9

Ajoutez la fonction `caesarDecypher` qui prend en paramètre un caractère `character` et qui retourne le caractère déchiffré selon l'algorithme de César (soit avec le décalage opposé : -3).

EXERCICE 10

Ajoutez la fonction `caesarDecypherArray` qui prend en paramètre une taille `size` et un tableau de caractère `charArray` et qui modifie chacune des valeurs du tableau en la remplaçant par le caractère déchiffré selon l'algorithme de César (soit avec le décalage opposé : -3).

EXERCICE 11

Ajoutez la fonction `showCharArray` qui prend en paramètre une taille `size` et un tableau de caractères associé `charArray`) et qui affiche le contenu du tableau à l'écran.

EXERCICE 12

À l'aide des fonctions que vous venez de coder, modifiez votre fonction `main` pour :

- Afficher le contenu du tableau, précédé de la phrase `"Original: "`.
- Chiffrer les caractères de votre tableau (vous remplacerez chaque caractère du tableau par sa valeur chiffrée).
- Afficher le contenu chiffré du tableau, précédé de la phrase `"Caesar Cypher: "`.
- Dechiffrer les caractères de votre tableau (vous remplacerez chaque caractère du tableau par sa valeur déchiffrée).
- Afficher le contenu déchiffré du tableau (vous devez retomber sur votre contenu de départ), précédé de la phrase `"Caesar Decypher: "`.

Aidez vous de l'affichage ci-dessous pour savoir quoi afficher.

```
Size of the array?
Please enter a value greater than 0:
-2
Please enter a value greater than 0:
3
Please enter a character
a
Please enter a character
b
Please enter a character
c
Original: abc
Caesar Cypher: def
Caesar Decypher: abc
```

EXERCICE 13

Soumettez votre code à l'autocorrect (dans la bonne section). Vos fonctions doivent avoir des prototypes corrects et faire **exactement** ce qui est attendu d'elles.

QUESTION 16 (*Défis*[2])

Que signifie `Wkdw*v#doo#iron#$$$` ?

Mémo

« Structures de données : variables & tableaux. »
« Algorithmes importants : parcours, recherche et tri. »
« Un tableau commence toujours par l'indice 0. »

2. Les questions défis sont des questions optionnelles que vous pouvez faire à la maison pour approfondir le TD.