**Langage de programmation**

**HAUTE ÉCOLE DE NAMUR-LIÈGE-LUXEMBOURG**

**Bloc 1**

Atelier 4 – Tableaux

Objectifs

* Déclarer et utiliser un tableau
* Déclarer des chaînes de caractères et utiliser les fonctions associées
* Utiliser des tableaux comme paramètres de fonction
* Distinguer le passage par valeur du passage par adresse

[Introduction 1](#_Toc138417381)

[A. Pourquoi des tableaux ? 2](#_Toc138417382)

[B. Déclarer et utiliser des tableaux 5](#_Toc138417383)

[C. Chaînes de caractères 13](#_Toc138417384)

# Introduction

Dans cet atelier, vous allez apprendre à déclarer et manipuler des tableaux.

Dans ce document, plusieurs conventions sont utilisées :

* les mots gras désignent des termes de vocabulaire liés à l’**informatique en général**.
* les mots soulignés et gras désignent des termes de vocabulaire directement liés aux cours de **programmation**.
* le logo signifie que vous avez quelque chose à réaliser.
* le logo est associé aux cadres présentant certaines conventions.
* le logo est associé aux cadres présentant les éléments liés à la propreté/lisibilité du code (*clean code*).

# Pourquoi des tableaux ?

Pour mieux comprendre l'utilité des tableaux, voici une mise en situation : dans un programme, vous pourriez avoir besoin de conserver plusieurs valeurs simultanément en mémoire de manière à les utiliser ultérieurement.

Par exemple, dans le cadre d’un concours de bowling, on veut garder les 5 derniers scores de chaque participant pour calculer leur moyenne. Ensuite, en fonction de cette moyenne, on désire afficher le score qui s'en rapproche le plus.

La seule solution dont vous disposez à l’heure actuelle consiste à déclarer 5 variables (appelées par exemple score1, score2, score3, etc.), à obtenir leur valeur et à calculer la moyenne. Vous devez ensuite déterminer l’écart de chaque score par rapport à cette moyenne et déterminer l’écart minimum de manière à trouver le score le plus proche de la moyenne.

Le programme en C correspondant à cette démarche est le suivant.

#include <math.h>

#include <stdio.h>

#define NB\_SCORES 5

void main(void) {

int score1, score2, score3, score4, score5;

double ecart1, ecart2, ecart3, ecart4, ecart5;

double moyenne;

double ecartMin;

int numEcartMin;

// obtentions

printf("Score 1 : ");

scanf("%d", &score1);

printf("Score 2 : ");

scanf("%d", &score2);

printf("Score 3 : ");

scanf("%d", &score3);

printf("Score 4 : ");

scanf("%d", &score4);

printf("Score 5 : ");

scanf("%d", &score5);

// moyenne

moyenne = (score1 + score2 + score3 + score4 + score5) / NB\_SCORES;

// écarts

ecart1 = abs(moyenne - score1);

ecart2 = abs(moyenne - score2);

ecart3 = abs(moyenne - score3);

ecart4 = abs(moyenne - score4);

ecart5 = abs(moyenne - score5);

// écart minimum

ecartMin = ecart1;

numEcartMin = 1;

if(ecart2 < ecartMin) {

ecartMin = ecart2;

numEcartMin = 2;

}

if(ecart3 < ecartMin) {

ecartMin = ecart3;

numEcartMin = 3;

}

if(ecart4 < ecartMin) {

ecartMin = ecart4;

numEcartMin = 4;

}

if(ecart5 < ecartMin) {

ecartMin = ecart5;

numEcartMin = 5;

}

// score le plus proche de la moyenne

printf("Score le plus proche de la moyenne %.2f : ", moyenne);

switch(numEcartMin) {

case 1 :

print("%d", score1);

break;

case 2 :

print("%d", score2);

break;

case 3 :

print("%d", score3);

break;

case 4 :

print("%d", score4);

break;

case 5 :

print("%d", score5);

break;

}

}

Il est évident que cette façon de travailler est d’autant plus illisible et laborieuse que le nombre de valeurs à traiter est important.

De plus, si vous ne connaissez pas avec exactitude la quantité de valeurs à traiter, le programme devient encore plus complexe à réaliser.

C’est pourquoi la programmation permet de rassembler toutes ces variables en une seule, au sein de laquelle chaque valeur est désignée par un numéro. Il s'agit d'un **tableau**.

Comme vous l’avez vu en « Principes de programmation », un tableau est une structure de données, plus précisément une collection de données, qui, en C, sont

* **homogènes** (même type pour toutes les données),
* **à accès direct** (via l’indice),
* **consécutives** en mémoire,
* à allocation **statique** (par opposition à dynamique),
* de **taille fixe**.

Attention, dans d’autres langages, comme le Javascript, les tableaux sont hétérogènes, non consécutifs, alloués dynamiquement ou de taille variable.

Statique vs dynamique

La différence principale entre l'allocation statique et l'allocation dynamique de mémoire est la suivante : dans le cas de l’allocation statique une fois que la mémoire est allouée, la taille de la mémoire est fixe, on ne peut pas augmenter ou diminuer l’espace mémoire alloué ; dans le cas de l'allocation dynamique, même après que la mémoire soit allouée, sa taille peut être modifiée, augmentée ou diminuée.

La gestion dynamique de la mémoire est un sujet abordé en profondeur dans le cadre du cours de « Langage de programmation : avancé » au Q2.

# Déclarer et utiliser des tableaux

Comme toute variable en C, un tableau doit être déclaré avant son utilisation.

## Déclaration

Pour rappel, la **déclaration** est l’instruction qui permet de préciser le **type** de la variable ainsi que son **nom**. Dans le cas du tableau, c’est le **type d’élément** que le tableau est destiné à contenir qui est nécessaire. Pour préciser qu’il s’agit d’un tableau, le nom de celui-ci est suivi par des crochets [].

De plus, comme pour les variables scalaires vues précédemment (int, double, char, bool), l’allocation de mémoire se fait de façon **statique**. En effet, lors de la compilation, la taille à allouer lors de l’exécution du programme est déterminée sur base du type et, dans le cas du tableau, de sa taille. Il est donc nécessaire de préciser le nombre d’éléments qui doivent être associés au nom du tableau. Lors de la déclaration d’un tableau, vous devez toujours préciser la **taille du tableau** en l’écrivant entre les crochets au moyen d’un littéral entier ou d’une expression dont le résultat est une valeur entière.

Pour déclarer un tableau, il faut donc respecter la syntaxe suivante :

<element\_type> <array\_name>[<size>];

Il faut y remplacer

* <element\_type> par le type qu’auront les éléments du tableau,
* <array\_name> par le nom du tableau,
* <size> par la taille du tableau, c’est-à-dire le nombre d’éléments/cellules qu’il contient.

En mémoire centrale, les cellules sont en fait l’une à la suite de l’autre. On réserve <size> \* sizeof(<element\_type>) octets consécutifs.

Dans l’exemple ci-dessus, il était question de 5 scores ainsi que de 5 écarts. Au lieu de passer par 5 variables pour chacun, il est possible de déclarer un tableau pour les scores, de taille 5, et un tableau pour les écarts, de taille 5 également. Voici les instructions permettant de déclarer ces deux tableaux (attention, aucune valeur n’est encore mémorisée) :

int scores[5];

double ecarts[5];

Attention, les espaces mémoire alloués à ces tableaux ne sont pas identiques puisque les scores sont des entiers (int 🡪 32 bits) et les écarts sont des réels (double 🡪 64 bits).

scores

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ... | ... | ... | ... | ... |

ecarts

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ... | ... | ... | ... | ... |

La déclaration d’un tableau de caractères contenant les voyelles s’écrit comme ci-dessous.

char voyelles[6];

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| voyelles |  |  |  |  |  |

À nouveau, l’espace mémoire alloué à ce tableau est lié à la place nécessaire pour encoder un caractère (char 🡪 8 bits)

Pour vous en rendre compte, dans un nouveau fichier source taillesTableaux.c, écrivez les instructions qui suivent. Notez que le format d’affichage %p permet d’afficher une adresse.

void main(void) {

char voyelles[6];

int scores[5];

double ecarts[5];

printf("Adresse de la premiere cellule de voyelles: %p\n", voyelles);

printf("Taille de voyelles en octets : %d\n", sizeof(voyelles));

printf("Adresse de la premiere cellule de scores : %p\n", scores);

printf("Taille de scores en octets: %d\n\n", sizeof(scores));

printf("Adresse de la premiere cellule de ecarts : %p\n", ecarts);

printf("Taille de ecarts en octets: %d\n", sizeof(ecarts));

}

Les tailles affichées devraient être 6 octets pour voyelles, 20 octets pour scores et 40 octets pour ecarts. Avez-vous bien compris pourquoi ? N’hésitez pas à faire un dessin sur une feuille, avec un crayon…

sizeof

L'opérateur unaire sizeof donne la taille en octets de son opérande. L'opérande peut être un type ou une expression. L'opérateur sizeof peut être appliqué à n’importe quel type de données.

Quand il s’agit d’une expression, l’expression est d’abord évaluée et sizeof donne la taille qui en résulte.

La valeur d'une opération sizeof est de type size\_t (un type entier non signé défini dans le fichier d'entête standard <stddef.h>).

## Initialisation lors de la déclaration

En C, il est possible d’initialiser un tableau sans passer par une boucle. Pour cela, vous devez lui affecter au moins une valeur lors de sa déclaration. Ces valeurs doivent correspondre au type des éléments du tableau, être séparées par des virgules , et le tout doit être entouré par des accolades {}.

Voici comment les tableaux scores et ecarts pourraient être initialisés.

int scores[5] = {256, 232, 245, 227, 214};

double ecarts[5] = {21.2, 2.8, 10.2, 7.8, 19.8};

En mémoire, chaque cellule se voit affecter la valeur qui lui correspond.

scores

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 256 | 232 | 245 | 227 | 214 |

ecarts

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 21.2 | 2.8 | 10.2 | 7.8 | 19.8 |

Pour initialiser le tableau des voyelles, vous pouvez procéder comme ci-dessous.

char voyelles[6] = {'a', 'e', 'i', 'o', 'u', 'y'};

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| a  voyelles | e | i | o | u | y |

Quelques remarques complémentaires au niveau de la syntaxe sont à faire :

* Si vous omettez la taille de celui-ci, le compilateur déduit la taille sur base du nombre de valeurs initiales. Dans l’exemple suivant, le tableau nombres aura donc une taille de 5.

int nombres[] = {-2, 7, 12, -10, 40};

* Si vous précisez la taille mais que vous initialisez seulement les premières cellules du tableau et pas les suivantes, le compilateur crée un tableau de la taille demandée et initialise les valeurs non précisées au zéro du type correspondant.

int nombres[5] = {-2, 7};

Dans ce cas, l’initialisation effective est donc {-2, 7, 0, 0, 0}.

Attention, il n’est pas possible de laisser des « trous », en C !

Il s’agit de **sucre syntaxique**.

Sucre syntaxique

Éléments de la syntaxe d’un langage de programmation qui le rendent plus facile à lire et à écrire, sans en changer la sémantique.

Le fait que lors d’une initialisation partielle d’un tableau les cellules non initialisées soient mises au zéro du type est du sucre syntaxique, au même titre que les opérateurs unaires ++ et --.

N’oubliez pas d'entourer les littéraux de type char par des apostrophes ''.

char alerte[8] = {'S','O','S'};

Le zéro du type char est '\0', appelé le caractère de fin de chaîne. Dans ce cas, l’initialisation effective est donc {'S','O','S','\0','\0','\0','\0','\0'}.

Dans un nouveau fichier source tableaux.c, écrivez les instructions pour

* déclarer et initialisez un tableau contenant les 6 voyelles (en minuscules).
* déclarer un tableau permettant de mémoriser les 4 nombres composant une adresse IP et initialisez le avec l’adresse 127.0.0.1. Ces valeurs sont des entiers.
* déclarez et initialisez un tableau contenant le taux de réussite de chaque bloc de la section IG sous la forme d’un réel. Le taux de réussite est un réel correspondant au pourcentage de réussite, par exemple si 42 étudiants sur 128 ont réussi au bloc 1, le taux de réussite de ce bloc est de 32,8 %.

## Atteindre un élément précis du tableau

Le nombre qui, au sein d’un tableau, sert à repérer chaque valeur s’appelle l’**indice**.

Il est important de rappeler que, pour des raisons techniques, l’indice commence à 0. Un tableau comprenant N éléments voit son indice aller de 0 à N-1.

En toute généralité, il faut citer le nom du tableau dont vous désirez atteindre une valeur et le faire suivre des crochets [] avec l’indice de la cellule en question entre ceux-ci.

La syntaxe est donc la suivante.

<array\_name>[<indice>]

Il faut y remplacer

* <array\_name> par le nom du tableau,
* <indice> par l’indice (position dans le tableau moins 1) de la cellule.

Dans l’exemple ci-dessus, le deuxième score est donc dans la cellule scores[1].

scores

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 256 | 232 | 245 | 227 | 214 |

scores[1]

## Affichage et lecture

Comme vous le savez, il est possible d’afficher la valeur d’une variable et de lire une valeur pour l’assigner à une variable. La fonction printf permet d’afficher à l’écran alors que la fonction scanf\_s permet de lire ce qui est entré au clavier.

La seule différence avec ce que vous avez utilisé jusqu’ici, c’est-à-dire des valeurs de type scalaire, est le fait que la valeur est mémorisée dans une cellule du tableau.

La syntaxe à respecter reste donc la même que ce soit pour un affichage ou pour une lecture, il faut juste préciser l’indice de la cellule visée :

printf("<write\_format>", <array\_name>[<indice>]);

scanf\_s("<read\_format>", **&**<array\_name>[<indice>]);

Il faut y remplacer

* <write\_format> par une chaîne de caractères comportant le(s) format(s) d’affichage correspondant au type de la (des) variable(s) à afficher,
* <read\_format> par une chaîne de caractères comportant le(s) format(s) de lecture correspondant au type de la (des) valeur(s) à récupérer,
* <array\_name> par le nom du tableau,
* <indice> par l’indice de la cellule.

Pour afficher la valeur de la cellule d'indice i du tableau scores, il faut utiliser la même notation que celle présentée ci-dessus pour cibler la cellule en question. L’instruction est la suivante.

printf("%d", scores[i]);

Pour garnir la cellule d'indice i du tableau en obtenant une valeur de l’utilisateur, c’est à nouveau comme pour les variables de type scalaire.

scanf\_s("%d", &scores[i]);

Dans le même fichier source que précédemment, tableaux.c, écrivez, en utilisant les structures de contrôle les plus adéquates, les instructions pour…

* afficher chacune des voyelles en les séparant par un tiret. Ne mettez pas de tiret inutile !
* demander à l’utilisateur les 4 nombres d’une adresse IP et les mémoriser dans le tableau.
* déterminer le taux de réussite moyen de la section et l’afficher.

Dans un nouveau fichier bowling.c, déclarez le tableau scores comme montré ci-dessus. Ne l’initialisez pas ! Écrivez les instructions permettant d’afficher l’indice, l’adresse et la valeur de chacune des cellules. Pensez à nouveau à utiliser la structure de contrôle adéquate afin d’éviter la répétition de code.

Le programme doit ressembler à celui proposé ci-dessous.

#define NB\_SCORES 5

void main(void) {

int scores[NB\_SCORES];

// afficher l'indice, l'adresse et la valeur de chaque cellule

for (int i = 0; i < NB\_SCORES; i++) {

printf("Cellule d'indice %d - Adresse %p - Valeur : %d\n", i, &scores[i], scores[i]);

}

}

Observez l’adresse affichée pour chacun des éléments. Vous remarquez qu’elles sont en effet consécutives dans la mémoire, puisque séparées de 4 bytes, taille d’un entier !

Cependant, dans le programme ci-dessus, vous n’avez fait que déclarer le tableau, sans affecter de valeurs aux cellules. Les valeurs affichées sont donc le résultat de l’interprétation de ce qui se trouve actuellement en mémoire à ces adresses-là.

Complétez ce programme de façon à demander les 5 scores à l’utilisateurs et à les mémoriser dans le tableau scores. Ensuite, affichez ces valeurs et assurez-vous d’avoir bien tout compris jusqu’ici !

## Pourquoi les indices commencent-ils à 0 ?

Comme vous allez le voir plus en détail dans le cours de « Architecture des ordinateurs », toutes les valeurs qui sont utilisées dans un programme passent à un moment donné par la mémoire centrale (et même par les registres de votre processeur). Les valeurs qui sont mémorisées dans des variables subissent le même sort et sont également en mémoire centrale pendant un certain temps appelé **durée de vie**.

Une valeur en mémoire y est localisable par l’**adresse mémoire** à laquelle elle se trouve.

C’est lors de la première phase d’analyse du code, pendant la compilation, que la table des symboles est construite. Il existe donc à un endroit précis, dans l’espace de la mémoire associée au programme exécuté, une table qui fait le lien entre les noms de variables et l’adresse à laquelle la valeur de celle-ci est mémorisée. Le processus est assez complexe et n’est pas vu en détail, mais il est important de comprendre qu’à tout nom de variable est associé l’adresse de sa valeur en mémoire.

Le nom d’un tableau n’y échappe pas ! Mais un tableau contient plusieurs valeurs… Il est alors associé à la première valeur qu’il contient, c’est-à-dire à l’adresse de cette première valeur : scores = &scores[0].

scores

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 256 | 232 | 245 | 227 | 214 |

Ainsi, l’accès aux autres valeurs du tableau se fait via un déplacement qui se calcule sur base de cette adresse et donc du nom du tableau. Par exemple, pour atteindre la deuxième cellule du tableau scores, il faut lui ajouter une fois la taille du type de valeurs contenues dans le tableau. Dans ce cas, il s’agit d’entiers, donc la valeur de la deuxième cellule se trouve à l’adresse mémoire correspondant à l’adresse du début du tableau plus une fois la taille d’un entier : scores + 1 \* sizeof(int).

+ 1 \* 32 bits 🡺 scores[1]

scores

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 256 | 232 | 245 | 227 | 214 |

Comme vous l’avez deviné, l’indice correspond au nombre de cellules de déplacement, et la formule générale qui est utilisée par le compilateur pour déterminer l’adresse de la valeur visée est : <array\_name> + <indice> \* sizeof(<element\_type>).

scores

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 256 | 232 | 245 | 227 | 214 |

+ 3 \* 32 bits 🡺 scores[3]

Le fait que l’indice commence à 0 permet d’avoir une formule générale, que ce soit la première cellule ou tout autre cellule du tableau, sur base de l’indice.

## Constante symbolique

Avant de compiler le code, le préprocesseur fait un premier traitement. Le **préprocesseur** est un programme exécuté lors de la première phase de la compilation.

Il ajoute, supprime ou remplace certaines parties du textedans le fichier source selon des **directives** interprétées.

Ces directives commencent par le caractère **#** et se terminent par le passage à la ligne. Pour poursuivre sur plusieurs lignes, on place le caractère **\** en fin de ligne.

Ces instructions ont pour but :

* l'incorporation de fichiers source (#include)
* la définition de **constantes symboliques** et de **macros** (#define)
* la **compilation conditionnelle** (#if, #ifdef...)

Dans cet atelier, seule la notion de constante symbolique est abordée.

Le but de l’utilisation des constantes symboliques est deremplacer les constantes du code par un texte significatif.

La syntaxe utilisée est la suivante :

#define <const\_name> <value>

Il faut y remplacer

* <const\_name> par le nom de la constante,
* <value> par une valeur (au sens large).

Nom de constante

Comme pour une variable, le nom d’une constante est un nom que vous choisissez vous-même assez librement.

Au niveau syntaxe, les noms de constantes doivent être écrits en majuscules et s’ils sont composés de plusieurs mots, il est d’usage de les séparer par des \_.

Comme pour les variables, efforcez-vous de bien les choisir : de préférence assez courts, mais aussi explicites que possible, de manière à exprimer clairement ce que la constante est censée contenir.

D'autres notations existent ou peuvent être imposées en fonction d'un projet, des habitudes d'une entreprise, etc.

Cette directive remplace <const\_name> par <value> dans toute la suite du fichier.

#define SMILEY ":)"  
#define NB\_MOIS 24

À partir de maintenant, veillez à utiliser les constantes symboliques pour les éléments constants et notamment pour la taille des tableaux.

La définition des constantes symboliques se fait usuellement en début de fichier source, juste après les #include.

## Revenons-en au programme de départ…

L’objectif de départ était de garder les 5 derniers scores de chaque participant pour calculer leur moyenne et afficher le score qui s'en rapproche le plus.

Dans le fichier source bowling.c, essayez de reproduire la solution correspondant à ce qui vous a été proposé, mais en utilisant les tableaux !

Le programme devrait ressembler à celui proposé ci-dessous.

#include <math.h>

#include <stdio.h>

#define NB\_SCORES 5

void main(void) {

int scores[NB\_SCORES];

double ecarts[NB\_SCORES];

int somme;

double moyenne;

double ecartMin;

int numEcartMin;

// obtentions

for(int i = 0; i < NB\_SCORES; i++) {

printf("Score %d : ", i + 1);

scanf\_s("%d", &scores[i]);

}

// moyenne

somme = 0 ;

for(int i = 0; i < NB\_SCORES; i++) {

somme += scores[i];

}

moyenne = (double)somme / NB\_SCORES;

// écarts

for(int i = 0; i < NB\_SCORES; i++) {

ecarts[i] = abs(scores[i] - moyenne);

}

// écart minimum

ecartMin = ecarts[0];

numEcartMin = 1;

for(int i = 1; i < NB\_SCORES; i++) {

if(ecarts[i] < ecartMin) {

ecartMin = ecarts[i];

numEcartMin = i + 1;

}

}

// score le plus proche de la moyenne

printf("Score le plus proche de la moyenne %.2f : ", moyenne);

printf("%d", scores[numEcartMin - 1]);

}

# Chaînes de caractères

En Java, en C#, en Python…, il y a un type particulier pour représenter les chaînes de caractères. En C, qui est un langage de moins haut niveau, il n'existe pas de type spécifique pour les chaînes de caractères.

En C, une chaîne de caractères est mémorisée sous la forme d’un tableau de caractères. Des fonctions adaptées à celles-ci sont disponibles dans la bibliothèque string.h. Un atelier supplémentaire est mis à votre disposition pour approfondir leur utilisation.

## Tableau de caractères vs chaîne de caractères

La différence entre un tableau de caractères et une chaîne de caractères est que cette dernière comporte toujours le caractère '\0' en fin de chaine.

Par exemple, si on mémorise la liste des voyelles dans un tableau, celui-ci contient les 6 voyelles… sans plus.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| a  voyelles | e | i | o | u | y |

Le tableau voyelles doit donc avoir une taille permettant de mémoriser exactement ces 6 caractères. Pour le déclarer et l’initialiser, il faut donc écrire l’instruction suivante, comme vous l’avez fait pour un tableau d’entiers ou de réels.

char voyelles[6] = {'a', 'e', 'i', 'o', 'u', 'y'};

Si on mémorise le prénom « Dennis » dans un tableau, celui-ci occupe les 6 premières cellules, mais il faut ajouter une cellule permettant de détecter la fin de la chaîne de caractères. En effet, comme l’allocation de mémoire se fait de manière statique, on doit prévoir assez de place pour tout prénom et il se peut que la chaîne de caractères n’occupe pas toutes les cellules du tableau… Il est donc nécessaire de repérer la cellule qui suit le dernier caractère de la chaîne, et cela se fait au moyen du '\0'.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| D  prenom | e | n | n | i | s | \0 |

Le tableau prenom doit donc avoir une taille permettant de mémoriser au moins le prénom et le caractère de fin de chaîne, ici au moins 7 caractères.

Pour déclarer une chaîne de caractères, il faut procéder comme précédemment, mais en n’oubliant pas de prévoir une cellule en plus pour le '\0'. Si vous voulez déclarer et initialisez la chaîne de caractères, il est inutile de préciser chaque élément séparément comme montré ci-dessus. Les chaînes de caractères sont initialisées au moyen d’un littéral chaîne de caractères, entouré par des "".

char prenom[T\_PRENOM] = "Dennis";

Cette notation ajoute implicitement le '\0' en fin de chaîne dans le tableau. Remarquez également l’utilisation d’une constante symbolique T\_PRENOM pour la taille du prénom. Cela implique que cette constante a été définie en début de fichier source comme ci-dessous.

#define T\_PRENOM 7

Dans un nouveau fichier source chaines.c, déclarez une variable permettant de mémoriser une adresse mail de 80 caractères maximum et initialisez-la avec votre adresse mail.

Vous devriez avoir une instruction similaire à ce qui est proposé ci-dessous.

char adresseMail[81] = "dennis.ritchie@inventeur.org";

N’hésitez pas à définir à nouveau une constante symbolique pour le 81…

Affichez ensuite l’adresse mail mémorisée via l’instruction suivante :

printf("Adresse mail : %s", adresseMail);

## Affichage et lecture

Jusqu’ici vous avez appris à afficher des chaînes de caractères via des littéraux, mais sans passer par des variables. Or, comme montré ci-dessus, il se peut que vous ayez besoin d’afficher une chaîne de caractères mémorisée dans une variable. Il vous arrivera aussi de vouloir lire une chaîne de caractères entrées par l’utilisateur…

### printf et puts

Pour afficher une variable de type « chaîne de caractères », la fonction printf a besoin du nom de la variable à afficher, c’est-à-dire le nom du tableau qui contient la chaîne de caractères. Il est inutile, et syntaxiquement incorrect, de préciser qu’il s’agit d’un tableau… il n’y a donc aucun [] à prévoir ici. Voici la syntaxe à respecter :

printf("<write\_format>", <array\_name>);

Il faut y remplacer

* <write\_format> par une chaîne de caractères comportant éventuellement le(s) format(s) d’affichage correspondant au type de la (des) variable(s) à afficher,
* <array\_name> par le nom du tableau.

Pour afficher ou obtenir une chaîne de caractères, vous devez utiliser le format adéquat : %s.

Dans le fichier source chaines.c, déclarez la variable prenom pouvant contenir maximum 50 caractères et initialisez-la avec votre prénom. Affichez ensuite le message « Mon prenom est … .», en remplaçant les « … » par la valeur de la variable en question.

Vous devriez avoir les instructions suivantes.

#define T\_PRENOM 51 // en début de fichier

char prenom[T\_PRENOM] = "Dennis";

printf("Mon prenom est %s.", prenom);

Dans le cas où l’instruction ne sert qu’à afficher une chaîne de caractères suivie d’un passage à la ligne, sans l’inclure dans un message, vous pouvez utiliser la fonction puts.

Voici la syntaxe à respecter.

puts(<string>);

Il faut y remplacer <string> par une chaîne de caractères soit sous la forme d’un littéral, soit sous la forme d’une variable de type « chaîne de caractères ».

Dans le fichier source chaines.c, affichez votre prénom et, sur la ligne suivante, « Quel joli prénom ! ».

Vous devriez avoir écrit les instructions suivantes.

puts(prenom);

puts("Quel joli prenom !");

La fonction puts ne prend donc qu’un seul argument en entrée !

### scanf\_s et gets\_s

Buffer overflow

En toute généralité, un ***buffer overflow***, en français **dépassement de tampon** ou **débordement de tampon**, arrive lorsqu’un programme écrit à l'extérieur de l'espace mémoire qui a initialement été alloué au tampon.

Ce dépassement de mémoire peut écraser des informations nécessaires à la suite du programme ou même à d’autres programmes. Le comportement de l'ordinateur devient alors imprévisible.

En C, il est possible de provoquer un dépassement lors de la lecture d’une information… En effet, si on utilise des instructions non sécurisées, elles peuvent placer dans un espace mémoire, comme celui alloué à une chaîne de caractères, plus de données qu’il ne peut en contenir. Dans ce genre de situation, les données excédantes sont insérées en mémoire même si elles écrasent les données qui se situent juste après.

En programmation, le dépassement de tampon est considéré comme une faille de sécurité. C’est pour cette raison que les fonctions sécurisées, dont le nom se termine par \_s, ont été ajoutées depuis la version C11 de la norme C. Celles-ci imposent que vous précisiez le nombre maximum de caractères mémorisables dans la zone mémoire associée à la variable cible.

De nombreux articles et tutoriaux sont disponibles sur le web pour en apprendre plus sur les soucis de sécurité liés au dépassement de tampon, mais des notions que vous n’avez pas encore sont nécessaires à leur compréhension. Une partie de ces notions est vue dans le cadre du cours d’« Architecture des ordinateurs » …

Pour lire une valeur de type « chaîne de caractères » et la mémoriser dans une variable, la fonction scanf\_s a besoin de l’adresse de la variable à assigner, or le nom du tableau correspond à l’adresse de la première cellule de la chaîne de caractères, donc le nom du tableau qui contient la chaîne de caractères suffit. Voici la syntaxe à respecter.

scanf\_s("<read\_format>", <array\_name>, <max\_size>);

Il faut y remplacer

* <read\_format> par une chaîne de caractères comportant le(s) format(s) de lecture correspondant au type de la (des) valeur(s) à récupérer,
* <array\_name> par le nom du tableau, et
* <max\_size> par la taille du tableau dans lequel la chaîne de caractères est mémorisée.

Attention ! La fonction scanf\_s ne lit les caractères du tampon d’entrée que jusqu’au premier espace rencontré ! Le reste des caractères entrés sont laissés tels quels dans le tampon.

Dans le cas où l’instruction ne sert qu’à lire une chaîne de caractères ou que la chaîne contient des espaces, vous pouvez utiliser la fonction gets\_s.

Voici la syntaxe à respecter.

gets\_s(<array\_name>, <max\_size>);

Il faut y remplacer

* <array\_name> par le nom d’une variable de type « chaîne de caractères » , et
* <max\_size> par la taille du tableau dans lequel la chaîne de caractères est mémorisée.

La fonction gets\_s ne prend donc que deux arguments en entrée !

Dans le fichier source chaines.c, écrivez les instructions qui permettent de lire votre prénom à l’aide d’un scanf\_s et ensuite d’un gets\_s. Affichez à chaque fois la valeur obtenue.

Vous devriez avoir les instructions suivantes.

#define T\_PRENOM 51 // en début de fichier

char prenom[T\_PRENOM] = "Dennis";

printf("Prenom : ");

scanf\_s("%s", prenom, T\_PRENOM) ;

printf("Mon prenom est %s.\n", prenom);

getchar();

printf("Prenom : ");

gets\_s(prenom, T\_PRENOM);

printf("Mon prenom est %s.\n", prenom);