

# Chaines de caractères (et tableaux)

En C il n'y a pas de type String, les chaines de caractères sont simplement des tableaux de caractères. Sauf que puis ce que l'on ne sait pas combien de la longueur du tableau a été rempli, donc on met un caractère de fin de chaîne à la fin du tableau `\0`.

```
char ma_chaine[21] = "Hello World!\n";
```

H	e	l	l	o		W	o	r	l	d	!	\n	\0							
---	---	---	---	---	--	---	---	---	---	---	---	----	----	--	--	--	--	--	--	--

Il est très important de toujours vérifier l'input des utilisateur·ice·s car si la personne entre quelque chose de plus long que la taille du tableau cela peut être une faille de vulnérabilité (car cela peut mener à un BufferOverflow). C'est notamment arrivé au programme `sudo` sous linux. Si vous voulez en apprendre plus vous pouvez regarder [cette vidéo](#).

## Lecture des chaines de caractères

Il existe par exemple `gets()`, `scanf()` ou encore `fgets()` pour prendre un input de l'utilisateur·ice.

Cependant il ne faut **pas** utiliser `gets()` car il ne vérifie pas la taille des données (ce qui peut donc mener à un BufferOverflow). Il faut donc toujours utiliser `scanf` ou `getf`.

### scanf

Voici par exemple comment récupérer les max 20 premiers caractères d'un input (le reste sera ignoré).

```
scanf("%20[^\n]*c", ma_chaine);
```

Pour déconstruire un peu ce qu'il se passe ici :

- `%20` signifie que l'on prends les 20 premiers caractères
- `[^\n]` signifie que l'on arrête de prendre des caractères quand l'utilisateur·ice fait ENTER
- `%*c` signifie que l'on ignore le dernier caractère (le retour à la ligne `\n`)

Autre chose intéressante à noter ici, il n'y a pas de `&` devant le nom de la fonction contrairement à avant quand on récupérait des caractères ou nombres uniques. Cela est dû au fait que `&` sert à passer l'adresse d'une variable (le pointeur) et qu'un tableau (comme une chaîne de caractères) est déjà une adresse (pointeur).

## fgets

`fgets` fonctionne assez différemment de `scanf`, voici comment on peut faire quelque chose de similaire à l'exemple précédent en utilisant `fgets` :

```
fgets(ma_chaine, 20, stdin);
```

Attention cependant que `fgets` compte `\n` comme un caractère et l'inclus dans le résultat. Donc bien que la syntaxe de `fgets` soit plus simple, il faut mieux utiliser `scanf` car elle s'occupe du caractère `\n` toute seule.

## Affichage des chaînes de caractères

### puts

L'exemple suivant va afficher la chaîne de caractère en y ajoutant un retour à la ligne automatiquement à la fin (c'est tout l'intérêt du `puts`), c'est un peu comme le `System.out.println` en Java :

```
puts(ma_chaine);
```

### fputs

Cet exemple fonctionne de manière similaire du `puts` sauf qu'il n'ajoute pas de retour à la ligne. C'est un peu comme le `System.out.print` en Java.

```
fputs(ma_chaine, stdout);
```

### printf

`Printf` est surtout intéressant pour formater l'affichage (c'est l'équivalent du `System.out.printf` en Java).

```
printf("%s\n", ma_chaine);
```

## Autres fonctions

Il existe une librairie `string` en C permettant d'interagir plus facilement avec les chaînes de caractères. Attention cependant, il ne faut pas la confondre avec le type `String` en Java, car en C "string" n'est pas un type les chaînes de caractères sont simplement des tableaux de `char`

Pour importer la librairie `string`, il suffit d'ajouter la ligne suivante au dessus du fichier :

```
#include <string.h>
```

Maintenant voici une petite liste des fonctions les plus utiles de `string` :

Fonction	Explication
<code>strlen(ma_chaine)</code>	Compte le nombre de caractères de la chaine jusqu'au <code>\0</code>
<code>strcmp(chaine1, chaine2, n)</code>	Compare les n premiers caractères des chaines. Si les deux sont les même cela signifie que les deux sont identiques
<code>strcpy(dest, source, n)</code>	Copie les n premiers caractères de la source vers la destination (le <code>\0</code> n'est pas ajouté)
<code>sprintf(dest, "%d + %d", 4, 5)</code>	Fait comme <code>printf</code> sauf qu'à la place de l'afficher, il le stocke dans une variable. C'est comme le <code>String.format</code> en Java
<code>scanf(src, "%d + %d", &amp;a, &amp;b)</code>	Fait comme le <code>scanf</code> sauf qu'a la place de le demander depuis le stdin (standard input), il va le prendre depuis une chaine de caractère source
<code>memset(src, n, 0)</code>	Initialise les n premiers caractères de la chaine src avec le caractère mentioné (ici on remplace tout par <code>\0</code> )
<code>strchr(chaine, car)</code>	Recherche la première occurrence d'un caractère dans une chaine et retourne un pointeur vers celle-ci
<code>strstr(chaine, sous-chaine)</code>	Recherche la première occurrence d'une sous-chaine donnée dans une chaine et retourne un pointeur vers celle-ci.

## Exemple de manipulation de tableau/chaines

```

/* Stocker une chaine dans un tableau */
char ma_chaine[20+1] = "Hello, World!";

/* Accéder au 5e caractère de la chaine */
printf("Le 5e caractère est %c\n", ma_chaine[4]);

/* Modifier un caractère */
ma_chaine[4] = ' ';

/* On peut aussi mettre le \0 n'importe où pour couper une chaine */
ma_chaine[4] = '\0';
printf("La chaine est maintenant : %s\n", ma_chaine);

```

Revision #2

Created 1 October 2023 09:43:35 by SnowCode

Updated 6 January 2024 19:14:47 by SnowCode