

# Module Algorithmie et programmation

Tableaux statiques et chaînes de caractères

---

Pierre Künzli

Adapté des cours de Paul Albuquerque, Guido Bologna et Orestis Malaspinas

Qu'est-ce qu'un tableau statique ?

## Qu'est-ce qu'un tableau statique ?

- Une **liste** ou un **ensemble**
- d'éléments du **même type**
- alloués de façon **contigüe** (en bloc) en mémoire
- sur la **pile**
- dont la taille **ne peut pas** être changée.

# Remarques

- Les éléments d'un tableau sont accédés avec `[i]` où `i` est l'index de l'élément.
- Le premier élément du tableau à l'index `0` !
- Lorsqu'un tableau est déclaré, la taille de celui-ci doit toujours être spécifiée, sauf s'il est initialisé lors de sa déclaration.
- Un tableau local à une fonction ne doit **jamais être retourné** !

# Syntaxe

- Allocation ;

```
#define SIZE 10  
int entiers[] = {2, 1, 4, 5, 7}; // taille 5, initialisé  
int tab[3]; // taille 3, non initialisé  
float many_floats[SIZE]; // taille 10, non initialisé
```

- Les indices sont numérotés de 0 à taille-1;

```
int premier = entier[0]; // premier = 2  
int dernier = entier[4]; // dernier = 7
```

- Les tableaux sont **non-initialisés** par défaut;
- Les bornes ne sont **jamais** vérifiées.

```
int indetermine_1 = tab[1]; // undefined behavior  
int indetermine_2 = entiers[5]; // UB
```

# Remarques

- Depuis C99 possibilité d'avoir des tableaux dont la taille est *inconnue à la compilation*;

```
int size;  
scanf("%d", &size);  
char string[size];
```

# Remarques

- Depuis C99 possibilité d'avoir des tableaux dont la taille est *inconnue à la compilation*;

```
int size;  
scanf("%d", &size);  
char string[size];
```

- Considéré comme une mauvaise pratique : que se passe-t-il si `size == 1e9` ?
- Allocation sur la pile, pas fait pour allouer beaucoup de mémoire.
- On préfère utiliser l'allocation **dynamique** de mémoire pour ce genre de cas-là (spoiler du futur du cours).

# Initialisation

- Les variables ne sont **jamais** initialisées en C par défaut.
- Question : Que contient le tableau suivant ?

```
double tab[4];
```



# Initialisation

- Les variables ne sont **jamais** initialisées en C par défaut.
- Question : Que contient le tableau suivant ?

```
double tab[4];
```

- Réponse : On en sait absolument rien !
- Comment initialiser un tableau ?

# Initialisation

- Les variables ne sont **jamais** initialisées en C par défaut.
- Question : Que contient le tableau suivant ?

```
double tab[4];
```

- Réponse : On en sait absolument rien !
- Comment initialiser un tableau ?

```
#define SIZE 10
double tab[SIZE];
for (int i = 0; i < SIZE; ++i) {
    tab[i] = rand() / (double)RAND_MAX * 10.0 - 5.0;
    // tab[i] contient un double dans [-5;5]
}
```

## Itérer sur les éléments d'un tableau

```
int x[10];  
for (int i = 0; i < 10; ++i) {  
    x[i] = 0;  
}  
  
int j = 0;  
while (j < 10) {  
    x[j] = 1;  
    j += 1;  
}  
  
int j = 0;  
do {  
    x[j] = -1;  
    j += 1;  
} while (j < 9)
```

# Recherche du minimum dans un tableau

## Problématique

Trouver la valeur minimale contenue dans un tableau et l'indice de l'élément le plus petit.

**Écrire un pseudo-code résolvant ce problème (4 min)**

**Postez votre proposition sur le chat du cours**

<https://cyberlearn.hes-so.ch/course/view.php?id=7788>

**Code algo2022**

# Recherche du minimum dans un tableau

## Problématique

Trouver la valeur minimale contenue dans un tableau et l'indice de l'élément le plus petit.

**Écrire un pseudo-code résolvant ce problème (4 min)**

**Postez votre proposition sur le chat du cours**

<https://cyberlearn.hes-so.ch/course/view.php?id=7788>

## Code algo2022

```
index = 0
min    = tab[0]
for i in [1; SIZE] {
    if min > tab[i] {
        min = tab[i]
        index = i
    }
}
```

# Recherche du minimum dans un tableau

**Implémenter ce bout de code en C (4min)**

# Recherche du minimum dans un tableau

**Implémenter ce bout de code en C (4min)**

```
int index = 0;
float min = tab[0];
for (int i = 1; i < SIZE; ++i) {
    if min > tab[i] {
        min = tab[i];
        index = i;
    }
}
```

# Représentation des tableaux en mémoire

La mémoire est :

- ... contigüe,
- ... accessible très rapidement

Exemple :

```
char tab[4] = {79, 91, 100, 88};
```

char	79	91	100	88	...	...
addr	2000	2001	2002	2003	...	...

Qu'est-ce que tab ?

```
tab;           // 2000, l'adress du 1er élément  
&tab[0];      // 2000 == tab  
tab[0];       // 79  
sizeof(tab);  // 4
```



# Les tableaux comme arguments de fonctions

- Un tableau en argument est le pointeur vers sa première case.
- Pas moyen de connaître sa taille : `sizeof()` inutile.
- Toujours spécifier la taille d'un tableau passé en argument.

```
void foo(int tab[]) { // équivalent à int *tab
    // que vaut sizeof(tab)?
    for (int i = 0; i < ?; ++i) { // taille?
        printf("tab[%d] = %d\n", i, tab[i]);
    }
}

void bar(int n, int tab[n]) { // [n] optionnel
    for (int i = 0; i < n; ++i) {
        printf("tab[%d] = %d\n", i, tab[i]);
    }
}
```

## Quels sont les bugs dans ce code ?

```
#include <stdio.h>

int main(void) {
    char i;
    char a1[] = { 100,200,300,400,500 };
    char a2[] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
    a2[10] = 42;

    for (i = 0; i < 5; i++) {
        printf("a1[%d] = %d\n", i, a1[i]);
    }

    return 0;
}
```

## Quels sont les bugs dans ce code ?

```
#include <stdio.h>

int main(void) {
    char i;
    // 200, ..., 500 char overflow
    char a1[] = { 100,200,300,400,500 };
    char a2[] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
    a2[10] = 42; // [10] out of bounds

    for (i = 0; i < 5; i++) {
        printf("a1[%d] = %d\n", i, a1[i]);
    }

    return 0;
}
```

# Tri par sélection

## **Problématique**

Trier un tableau par ordre croissant. Comment faire ?

# Tri par sélection

## Problématique

Trier un tableau par ordre croissant. Comment faire ?

## Idée d'algorithme

```
ind = 0
boucle (ind < SIZE-1) {
    Trouver le minimum du tableau, tab_min[ind:SIZE].
    Échanger tab_min avec tab[ind]
    ind += 1
}
```

Plein d'autres existent.

**A implémenter dans la série d'exercices**

# Un type de tableau particulier

## Les chaînes de caractères

```
string = tableau + char + magie noire
```

# Le type `char`

- Le type `char` est utilisé pour représenter un caractère.
- C'est un entier 8 bits signé.
- En particulier :

- Écrire

```
char c = 'A';
```

- Est équivalent à :

```
char c = 65;
```

- Les fonctions d'affichage interprètent le nombre comme sa valeur ASCII.

# Chaînes de caractères (strings)

- Chaîne de caractère == tableau de caractères **terminé par la valeur** `'\0'` ou `0`.

## Exemple

```
char *str  = "HELLO !";  
char str[] = "HELLO !";
```

Est représenté par

char	H	E	L	L	O		!	\0
ASCII	72	69	76	76	79	32	33	0



# Chaînes de caractères (strings)

- Chaîne de caractère == tableau de caractères **terminé par la valeur** `'\0'` ou `0`.

## Exemple

```
char *str = "HELLO !";  
char str[] = "HELLO !";
```

Est représenté par

char	H	E	L	L	O		!	\0
ASCII	72	69	76	76	79	32	33	0

A quoi sert le `\0` ?

# Chaînes de caractères (strings)

- Chaîne de caractère == tableau de caractères **terminé par la valeur** `'\0'` ou `0`.

## Exemple

```
char *str = "HELLO !";  
char str[] = "HELLO !";
```

Est représenté par

char	H	E	L	L	O		!	\0
ASCII	72	69	76	76	79	32	33	0

## A quoi sert le `\0` ?

Permet de connaître la fin de la chaîne de caractères (pas le cas des autres sortes de tableaux).

# Syntaxe

```
char name[5];  
name[0] = 'P'; // = 70;  
name[1] = 'a'; // = 97;  
name[2] = 'u'; // = 117;  
name[3] = 'l'; // = 108;  
name[4] = '\\0'; // = 0;  
char name[] = {'P', 'a', 'u', 'l', '\\0'};  
char name[5] = "Paul";  
char name[] = "Paul";  
char name[100] = "Paul is not 100 characters long.";
```

## Remarque

```
char str[] = "abc" != char str[] = {'a', 'b', 'c'}  
char str[] = "abc";  
char str2[] = {'a', 'b', 'c'};  
// Que vaut sizeof(str) ? et sizeof(str2) ?
```

# Fonctions

- Il existe une grande quantités de fonction pour la manipulation de chaînes de caractères dans `string.h`.
- Fonctions principales :

```
// longueur de la chaîne (sans le \0)  
size_t strlen(char *str);  
// copie jusqu'à un \0  
char *strcpy(char *dest, const char *src);  
    // copie len char  
char *strncpy(char *dest, const char *src, size_t len);  
// compare len chars  
int strncmp(char *str1, char *str2, size_t len);  
// compare jusqu'à un \0  
int strcmp(char *str1, char *str2);
```

- Pour avoir la liste complète : man `string`.

# Fonctions

- Il existe une grande quantité de fonction pour la manipulation de chaînes de caractères dans `string.h`.
- Fonctions principales :

```
// longueur de la chaîne (sans le \0)
size_t strlen(char *str);
// copie jusqu'à un \0
char *strcpy(char *dest, const char *src);
// copie len char
char *strncpy(char *dest, const char *src, size_t len);
// compare len chars
int strncmp(char *str1, char *str2, size_t len);
// compare jusqu'à un \0
int strcmp(char *str1, char *str2);
```

- Pour avoir la liste complète : man `string`.

**Quels problèmes peuvent se produire avec `strlen`, `strcpy`, `strcmp` ?**

# Les anagrammes

## Définition

Deux mots sont des anagrammes l'un de l'autre quand ils contiennent les mêmes lettres mais dans un ordre différent.

## Exemple

t	u	t	u	t	\0
t	u	t	t	u	\0

**Problème :** Trouvez un algorithme pour déterminer si deux mots sont des anagrammes.

# Les anagrammes

Il suffit de :

1. Trier les deux mots.
2. Vérifier s'ils contiennent les mêmes lettres.

Une implémentations

On suppose qu'on sait trier un tableau

```
int main() { // pseudo C
    tri(mot1);
    tri(mot2);
    if egalite(mot1, mot2) {
        // anagrammes
    } else {
        // pas anagrammes
    }
}
```

# Les palindromes

Mot qui se lit pareil de droite à gauche que de gauche à droite :



# Les palindromes

Mot qui se lit pareil de droite à gauche que de gauche à droite :

- rotor, kayak, ressasser, ...

**Problème : proposer un algorithme pour détecter un palindrome**

# Les palindromes

Mot qui se lit pareil de droite à gauche que de gauche à droite :

- rotor, kayak, ressasser, ...

**Problème : proposer un algorithme pour détecter un palindrome**

## Solution 1

```
while (first_index < last_index {  
    if (mot[first_index] != mot [last_index]) {  
        return false;  
    }  
    first_index += 1;  
    last_index -= 1;  
}  
return true;
```

**Autre idée ?**

## Solution 2

```
mot_tmp = revert(mot)
return egalite(mot, mot_tmp)
```

Nécessite une fonction `revert` qui inverse une chaîne de caractères.