R101 — Introduction au développement

Initiation à la programmation en langage C
Tableaux

Clément Pagès
clement.pages@u-pec.fr

Université Paris-Est Créteil
BUT Informatique 1re année

2025-2026

Sommaire

1 Tableaux

2 Pointeurs

3 Lien entre pointeurs et tableaux

Rappels sur les types

Types natifs en C

- En C, les variables sont typées (`int`, `unsigned int`, `long`, `float`, `double`, etc). Le type de la variable détermine les limites des valeurs qu'elle peut prendre.
- Une variable est un emplacement mémoire, donc la taille dépend du type de la variable.
- Le type `char` est en général réservé pour les caractères (ASCII ou Unicode, selon la plateforme).

Rappels sur les types

Types natifs en C

- En C, les variables sont typées (`int`, `unsigned int`, `long`, `float`, `double`, etc). Le type de la variable détermine les limites des valeurs qu'elle peut prendre.
- Une variable est un emplacement mémoire, donc la taille dépend du type de la variable.
- Le type `char` est en général réservé pour les caractères (ASCII ou Unicode, selon la plateforme).

Définition : tableaux

Un tableau en C est une variable composées de données de même type, stockées de manière contiguë en mémoire.

Manipulation des tableaux

Déclaration et initialisation d'un tableau

```c
int tab1[4];
float tab2[5] = {5.4, 2.6, -45.78, -4.01, 0.0};
char message[] = {'B', 'o', 'n', 'j', 'o', 'u', 'r'};
```

Manipulation des tableaux

Déclaration et initialisation d'un tableau

```c
int tab1[4];
float tab2[5] = {5.4, 2.6, -45.78, -4.01, 0.0};
char message[] = {'B', 'o', 'n', 'j', 'o', 'u', 'r'};
```

- Le premier élément porte l'indice zéro
  - `tab2[0] = 5.4`
  - `message[6] = 'r'`
- On accède à chaque élément du tableau par son indice : `tab2[3] = -4.01`.
- Un tableau se parcourt avec une boucle `for`. Cela nécessite de connaître la taille du tableau :

Manipulation des tableaux

Déclaration et initialisation d'un tableau

```c
int tab1[4];
float tab2[5] = {5.4, 2.6, -45.78, -4.01, 0.0};
char message[] = {'B', 'o', 'n', 'j', 'o', 'u', 'r'};
```

- Le premier élément porte l'indice zéro
  - `tab2[0] = 5.4`
  - `message[6] = 'r'`
- On accède à chaque élément du tableau par son indice : `tab2[3] = -4.01`.
- Un tableau se parcourt avec une boucle `for`. Cela nécessite de connaître la taille du tableau :

```c
for(int i=0, i<5 ; i++)
    printf("%.2f\n", tab2[i]);
```

```
5.40
2.60
-45.78
-4.01
0.00
```

Plus de détails sur les tailles des tableaux

L'opérateur `sizeof`

- L'opérateur `sizeof` renvoie la taille, en octets, occupée par une variable.
- Sur ma machine, `sizeof(int) = 4`.

Plus de détails sur les tailles des tableaux

L'opérateur `sizeof`

- L'opérateur `sizeof` renvoie la taille, en octets, occupée par une variable.
- Sur ma machine, `sizeof(int) = 4`.

Opérateur `sizeof` et tableaux

L'opérateur `sizeof(tab)` renvoie l'espace mémoire occupé par la totalité du tableau.

```c
int tab[3];
sizeof(int); // 4
sizeof(tab); // 12 (4×3)
sizeof(tab[1]); // 4
```

Plus de détails sur les tailles des tableaux

L'opérateur `sizeof`

- L'opérateur `sizeof` renvoie la taille, en octets, occupée par une variable.
- Sur ma machine, `sizeof(int) = 4`.

Opérateur `sizeof` et tableaux

L'opérateur `sizeof(tab)` renvoie l'espace mémoire occupé par la totalité du tableau.

```c
int tab[3];
sizeof(int); // 4
sizeof(tab); // 12 (4×3)
sizeof(tab[1]); // 4
```

Retrouver le nombre d'éléments d'un tableau

- Un tableau contient des éléments qui sont tous du même type.
- `nbElements = sizeof(tab)/sizeof(tab[0]);`

Sommaire

1 Tableaux

2 Pointeurs

3 Lien entre pointeurs et tableaux

Adresse d'une variable

- Une variable `int x;` est un emplacement mémoire dans lequel on peut stocker de l'information (ici un nombre entier).

Adresse d'une variable

- Une variable `int x;` est un emplacement mémoire dans lequel on peut stocker de l'information (ici un nombre entier).
- Cette information a une adresse en mémoire.

Adresse d'une variable

- Une variable `int x;` est un emplacement mémoire dans lequel on peut stocker de l'information (ici un nombre entier).
- Cette information a une adresse en mémoire.
- On accède à l'adresse de `x` via `&x`.

Adresse d'une variable

- Une variable `int x;` est un emplacement mémoire dans lequel on peut stocker de l'information (ici un nombre entier).
- Cette information a une adresse en mémoire.
- On accède à l'adresse de `x` via `&x`.

Adresse d'une variable

- Une variable `int x;` est un emplacement mémoire dans lequel on peut stocker de l'information (ici un nombre entier).
- Cette information a une adresse en mémoire.
- On accède à l'adresse de `x` via `&x`.

```c
int x = 3; // Création et affectationn de la variable x

printf("Valeur de x :    %d\n", x); // Affiche 3
printf("Adresse de x x : %p", &x);  // Affiche 0x7fff53edeb8c
```

Variable... de type pointeur !

- Il est possible de stocker l'adresse d'une variable...

Variable... de type pointeur !

- Il est possible de stocker l'adresse d'une variable...

Variable... de type pointeur !

- Il est possible de stocker l'adresse d'une variable...dans une variable.

Variable... de type pointeur !

- Il est possible de stocker l'adresse d'une variable...dans une variable.
- La variable obtenue est appelée *pointeur*.

Variable... de type pointeur !

- Il est possible de stocker l'adresse d'une variable...dans une variable.
- La variable obtenue est appelée *pointeur*.

Variable... de type pointeur !

- Il est possible de stocker l'adresse d'une variable...dans une variable.
- La variable obtenue est appelée *pointeur*.

```c
int x = 3; // Création et affectationn de la variable x
int* p = NULL; // Pointeur vers un int
               // Pour l'instant, ne pointe vers rien.
p = &x; // Affectation de p avec l'adresse de x

printf("Valeur de x :    %d\n", x); // Affiche 3
printf("adresse de x x : %p", p);  // Affiche 0x7fff53edeb8c
```

Variable... de type pointeur !

- Il est possible de stocker l'adresse d'une variable...dans une variable.
- La variable obtenue est appelée *pointeur*.

```c
int x = 3; // Création et affectationn de la variable x
int* p = NULL; // Pointeur vers un int
               // Pour l'instant, ne pointe vers rien.
p = &x; // Affectation de p avec l'adresse de x

printf("Valeur de x :    %d\n", x); // Affiche 3
printf("adresse de x x : %p", p);  // Affiche 0x7fff53edeb8c
```

Défintion : pointeur

On dit que p est un pointeur sur x ou que p pointe vers x.

Variable... de type pointeur !

- Il est possible de stocker l'adresse d'une variable...dans une variable.
- La variable obtenue est appelée *pointeur*.

```c
int x = 3; // Création et affectationn de la variable x
int* p = NULL; // Pointeur vers un int
               // Pour l'instant, ne pointe vers rien.
p = &x; // Affectation de p avec l'adresse de x

printf("Valeur de x :    %d\n", x); // Affiche 3
printf("adresse de x x : %p", p);  // Affiche 0x7fff53edeb8c
```

Défintion : pointeur

On dit que p est un pointeur sur x ou que p pointe vers x.

Initialiation des pointeurs

Toujouurs initialiser un pointeur.

Pointeur et variable pointée

```c
printf("%d", x);  // Affiche 3
printf("%p", &x); // Affiche l'adresse de x
printf("%p", p);  // Affiche la valeur de p, i.e. l'adresse de x
printf("%d", *p); // Affiche 3 : valeur de la variable pointée
printf("%p", &p); // Que fait cette ligne ?
```

Pointeur et variable pointée

```c
printf("%d", x);  // Affiche 3
printf("%p", &x); // Affiche l'adresse de x
printf("%p", p);  // Affiche la valeur de p, i.e. l'adresse de x
printf("%d", *p); // Affiche 3 : valeur de la variable pointée
printf("%p", &p); // Que fait cette ligne ?
```

Le double sens de l'opérateur *

L'opérateur * a deux signification duales !

- Créer un pointeur vers une variable : `int* p = &x;`.
- Accéder à une valeur pointée : `*p`.

Pointeur et variable pointée

```c
printf("%d", x);  // Affiche 3
printf("%p", &x); // Affiche l'adresse de x
printf("%p", p);  // Affiche la valeur de p, i.e. l'adresse de x
printf("%d", *p); // Affiche 3 : valeur de la variable pointée
printf("%p", &p); // Que fait cette ligne ?
```

Le double sens de l'opérateur *

L'opérateur * a deux signification duales !

- Créer un pointeur vers une variable : `int* p = &x;`.
- Accéder à une valeur pointée : `*p`.

Quel est l'affichage du code suivant ?

```c
int x = 3, *p = &x;
*p = 5;
printf("%d", x);
```

Sommaire

1 Tableaux

2 Pointeurs

3 Lien entre pointeurs et tableaux

Adresse du premier élément

- Un tableau est un pointeur vers le premier élément.

Adresse du premier élément

- Un tableau est un pointeur vers le premier élément.

Exemple

```c
float tab[] = {-4.5, 2.1, 4};

printf("%p", tab); // Affiche l'adresse de tab[0]
printf("%f", tab); // Affiche une valeur n'ayant aucun sens
printf("%f", tab[0]); // Affiche -4.5
```

Adresse du premier élément

- Un tableau est un pointeur vers le premier élément.

Exemple

```c
float tab[] = {-4.5, 2.1, 4};

printf("%p", tab); // Affiche l'adresse de tab[0]
printf("%f", tab); // Affiche une valeur n'ayant aucun sens
printf("%f", tab[0]); // Affiche -4.5
```

- On a l'égalité (mathématique !) `tab` = `&(tab[0])`

Adresse du premier élément

- Un tableau est un pointeur vers le premier élément.

Exemple

```c
float tab[] = {-4.5, 2.1, 4};

printf("%p", tab); // Affiche l'adresse de tab[0]
printf("%f", tab); // Affiche une valeur n'ayant aucun sens
printf("%f", tab[0]); // Affiche -4.5
```

- On a l'égalité (mathématique !) `tab` = `&(tab[0])`
- Encore plus rigolo !

```c
float tab[] = {-4.5, 2.1, 4};

printf("%p", p); // Affiche l'adresse de tab[0]
p++; // On incrémente l'adresse : décalage d'indice
printf("%p", p); // Affiche l'adresse de tab[1]
printf("%f", *p); // Quel est l'affichage ? :-)
```

Tableaux de taille variable

- On souhaite créer des tableaux de taille variable ou dont la taille n'est pas connue à la compilation.
- Cas d'usage : saisie d'une liste de valeurs en quantité choisie par l'utilisateur·rice.

Tableaux de taille variable

- On souhaite créer des tableaux de taille variable ou dont la taille n'est pas connue à la compilation.
- Cas d'usage : saisie d'une liste de valeurs en quantité choisie par l'utilisateur·rice.

L'allocation dynamique de mémoire

On réserve de la mémoire manuellement, au moment de l'exécution.

Tableaux de taille variable

- On souhaite créer des tableaux de taille variable ou dont la taille n'est pas connue à la compilation.
- Cas d'usage : saisie d'une liste de valeurs en quantité choisie par l'utilisateur·rice.

L'allocation dynamique de mémoire

On réserve de la mémoire manuellement, au moment de l'exécution. Le système va renvoyer *l'adresse* de l'emplacement mémoire attribué.

Allocation dynamique

Fonction `malloc()`, `realloc()` et `free()`

- Utilise la bibliothèque `stdlib.h`.
- Allouer et libérer la mémoire :

Allocation dynamique

Fonction `malloc()`, `realloc()` et `free()`

- Utilise la bibliothèque `stdlib.h`.
- Allouer et libérer la mémoire :

```c
void* malloc(int tailleSouhaitee);
void* realloc(void* p_dejaReservee, int nouvelleTaille);
void free(void* p_adresseALiberer);
```

Allocation dynamique

Fonction `malloc()`, `realloc()` et `free()`

- Utilise la bibliothèque `stdlib.h`.
- Allouer et libérer la mémoire :

```c
void* malloc(int tailleSouhaitee);
void* realloc(void* p_dejaReservee, int nouvelleTaille);
void free(void* p_adresseALiberer);
```

Libérez la mémoire !

Toute mémoire allouée avec `malloc` ou `realloc` doit impérativement être libérée lorsqu'elle n'est plus utile.

Allocation dynamique

Exemple : tableau de taille variable

```c
#include <stdlib.h>
int main (void) {
    float* tab = NULL; // Future adresse pour notre tableau
    int taille = 1000; // On pourrait lire la valeur avec scanf

    // Allocation dynamique
    valeurs = (float*) malloc(taille * sizeof(float));

    // Vérification de l'allocation et remplissage du tableau
    if (valeurs == NULL) {
        printf("Echec de l'allocation\n");
    }
    else {
        for(int i=0; i<taille; i++) valeurs[i] = 0;
    }
    free(valeurs); // Libération de la mémoire

    return EXIT_SUCCESS;
}
```