Langage *C* : Allocation dynamique

Sébastien Jean

IUT de Valence
Département Informatique

v1.0, 26 novembre 2025

IUT Valence
Université Grenoble Alpes

Interlude : Création/clonage d'un projet Gitlab

- **Créer un projet** `AllocationDynamiqueC` sur Gitlab (avec un `README`)
- **Cloner** le projet depuis *VsCode* et ouvrir le dépôt
- **Modifier** le fichier `README.md` pour indiquer à quoi sert ce projet
- N.B. : pour chacun des exemples/exercices `ExX` suivant :
  - **Ecrire le programme** dans `ExX/src/main.c` et le compiler dans `ExX/build/ExX`
  - `Rédiger (si nécessaire) un jeu d'essai` dans un fichier `ExerciceX/Essai`

IUT Valence Université Grenoble Alpes

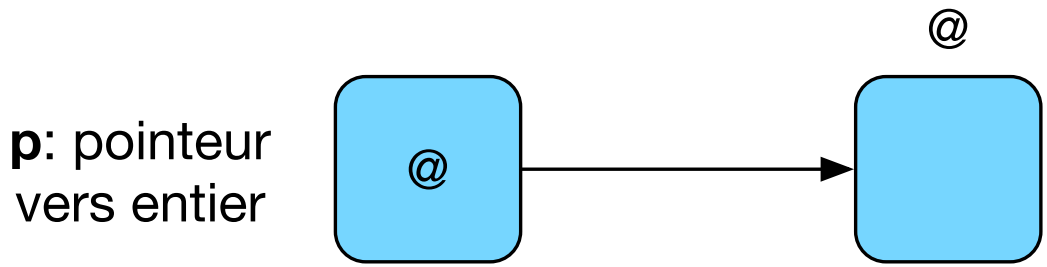
Allocation dynamique de mémoire (rappels)

- **Allocation dynamique**
  - Réservation de zone mémoire *à la volée*, sans variable

```
VARIABLE p : pointeur vers entier

p ← ALLOUER entier
```

- `ALLOUER` réserve une nouvelle zone mémoire de taille adaptée au stockage d'une valeur de type donné et **retourne l'adresse de début de la zone mémoire**



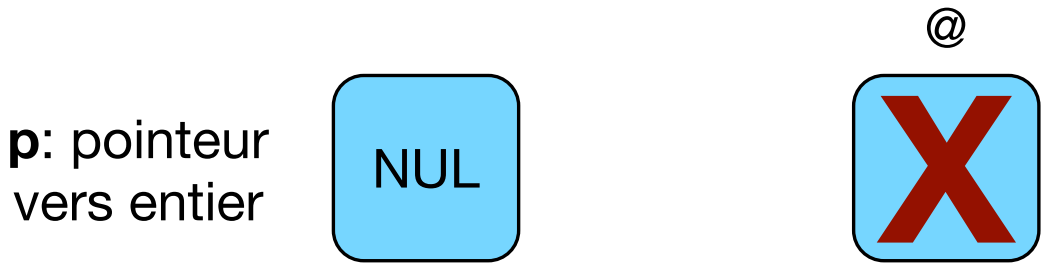
IUT Valence
Université Grenoble Alpes

Allocation dynamique de mémoire (rappels)

- **Libération de mémoire**
  - En théorie : pas de contrainte sur la quantité de mémoire disponible
  - En pratique : ressources limitées, judicieux d'économiser.

```
VARIABLE p : pointeur vers entier
p ← ALLOUER entier
...
LIBERER p
```

- `LIBERER` libère la zone mémoire située à partir de l'adresse contenue dans le pointeur et de la taille correspondant au type de valeur référencée par le pointeur



C : allocation dynamique

- Allocation suivant la syntaxe `malloc(taille)`
- Libération suivant la syntaxe `free(adresse)`

Code C

```c
int *i_ptr = (int *)malloc(sizeof(int));
...
free(i_ptr);
```

- `malloc` et `free` sont définies dans la librairie `stdlib` (à inclure)

IUT Valence Université Grenoble Alpes

C : allocation dynamique

- Signature des fonctions :
  - `void * malloc(size_t size);`
    - retourne un **pointeur générique** (`void *`) promu à l'utilisation
  - `void free(void *);`
    - prend en paramètre un pointeur générique (n'importe quel pointeur convient)
- Le pointeur `void *` permet de passer en paramètre ou retourner une **adresse sans indication de type**
  - Pour accéder à la donnée située à cette adresse, il faut **caster l'adresse** dans le type de pointeur voulu (`unsigned char *`, `int *`, ...)

IUT Valence Université Grenoble Alpes

Exercice

- Ecrire une fonction `main` qui :
  - Créé une variable entière `i`
  - Alloue dynamiquement un entier et stocke son adresse dans une variable `i_ptr`
  - Affiche les adresses des 2 variables et l'adresse de la zone mémoire allouée

IUT Valence
Université Grenoble Alpes

Exercice

Code C

```c
#include <stdio.h>
#include <stdlib.h>

int main() {

    int i = 1;
    printf("i :  %p\n", &i);

    int *i_ptr = (int *) malloc(sizeof(int));
    printf("i_ptr :  %p\n",& i_ptr);
    printf("i_ptr :  %p\n", i_ptr);

    return 0;
}
```

- N.B. Le cast de `void *` est implicite dans le cas d'une

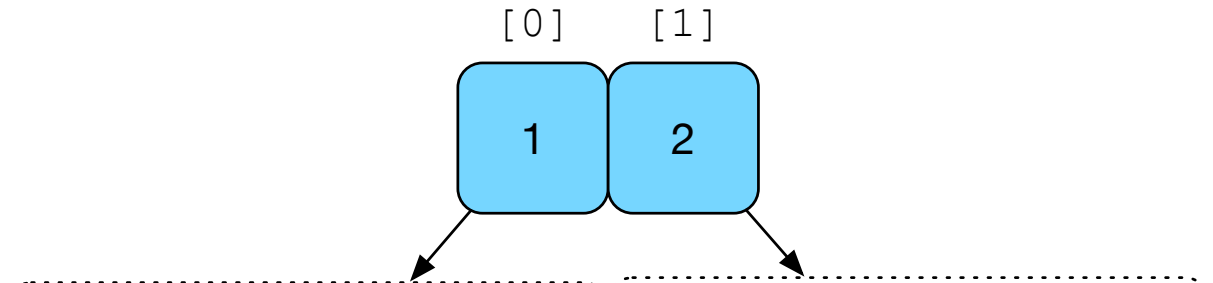
Dualité pointeur/tableau en C

- En C, un tableau est une **zone mémoire**, dont la **taille dépend du nombre de cases et du type des éléments**, et où les cases sont contigües et dans l'ordre
- Les cases du tableau sont représentées par des zones mémoire dont la **taille dépend du type des éléments**

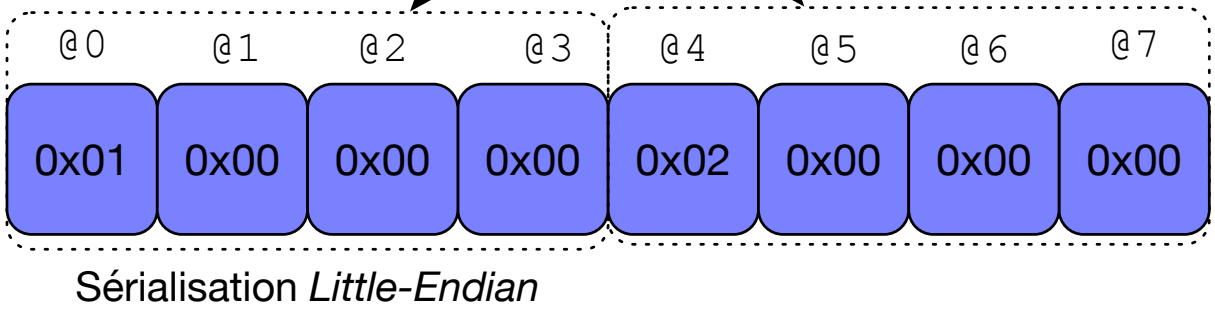
Déclaration/affectation

```
int t[2] = {1,2};
```

Représentation logique



Représentation mémoire

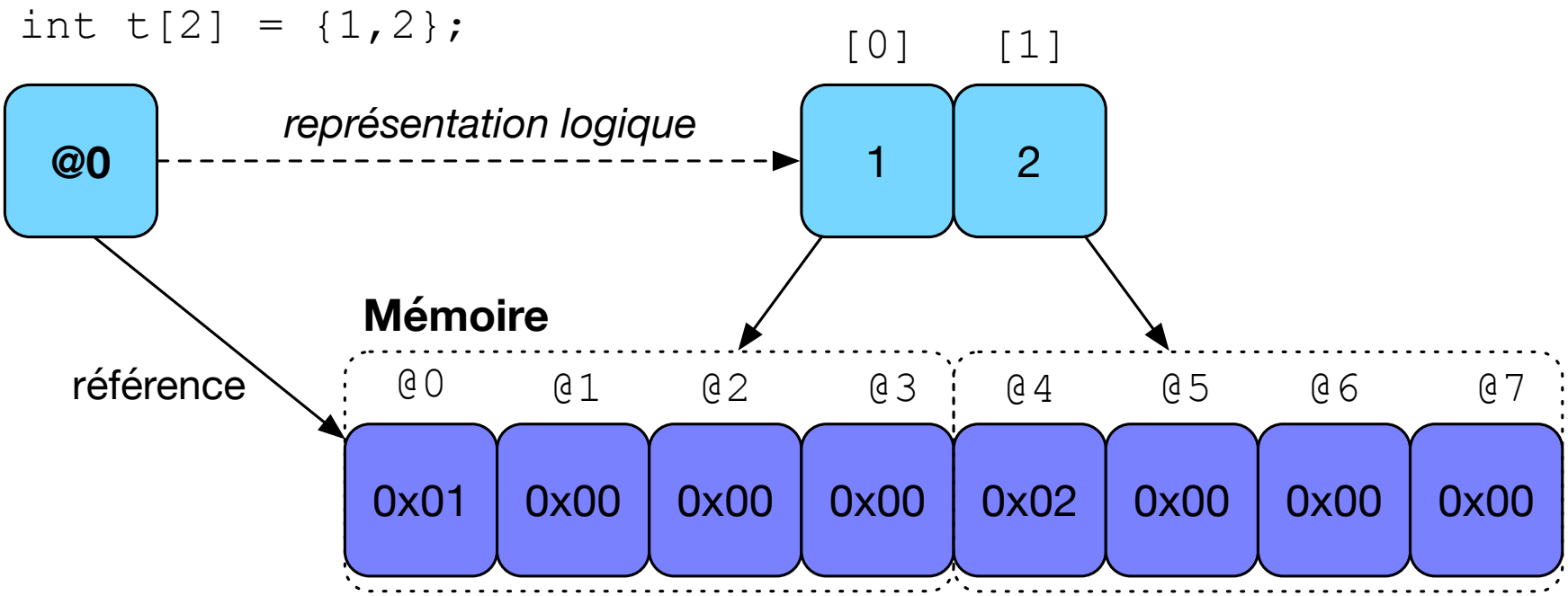


Sérialisation *Little-Endian*

IUT Valence Université Grenoble Alpes

Dualité pointeur/tableau en C

- En C, une variable de type tableau de `T` est en réalité **de type pointeur vers `T`**, contenant l'adresse de la première case du tableau



- `t` ≡ `&(t[0])`
- En C, on ne peut pas réaffecter une variable de type tableau (constante)

IUT Valence Université Grenoble Alpes

Dualité pointeur/tableau en C

- En C, on peut manipuler un tableau soit par arithmétique de pointeurs (*), soit avec l'accès indicé ([])
  - `t[i]` $\equiv$ `*(t + i)`
  - `&t[i]` $\equiv$ `(t + i)`

Code C

```c
int t[2] = {1, 2};

// équivalence [] / *

t[0] = -7;
*t = -7;

int j = t[1];
int k = *(t + 1);
```

Dualité pointeur/tableau en C

- On ne peut pas retourner de tableau de `T`, mais on peut **retourner un pointeur** vers `T`
  - La valeur de retour doit être stockée dans un pointeur, qui peut ensuite être manipulée et vue comme un tableau dont on ne connait pas la taille

Code C

```c
int * f(int n) {
    int *t = (int *) malloc(n*sizeof(int));
    return t;
}

int main() {
    int *t = f(5);
    t[0] = 1;
    ...
    return 0;
}
```

Dualité pointeur/tableau en C

- On peut passer un pointeur vers T à une fonction qui attend un paramètre tableau de T

Code C

```c
#include ...

void f(int t[], int n) { printf("%d\n", t[n]); }

int main() {
    int *t = (int *) malloc(10*sizeof(int));
    for (int i=0; i<10;i++) {t[i] = i;}
    f(t, 3);
    return 0;
}
```

Exercice

Enoncé du problème

On veut inverser les valeurs d'un tableau.

Spécification du problème

- **Donnée d'entrée** : t, tableau d'entiers (le tableau à inverser)
- **Donnée d'entrée** : n, entier (la taille de `t`)
- **Donnée de sortie** : r, tableau d'entiers (le tableau inversé)
- **Pré-condition** : `n` $\geq$ `1`
- **Post-condition** : `r` est un tableau de même taille que `t`, possédant les mêmes éléments mais dans l'ordre inverse

Signature de la fonction

- **inverser_tableau** (t : tableau d'entier, n : entier) : tableau d'entier

Université Grenoble Alpes

Exercice

- Ecrire la fonction et le `main` en :
  - Prenant en paramètre un pointeur au lieu d'un tableau
  - Retournant un nouveau tableau alloué dynamiquement
  - Effectuant le parcours en utilisant les 2 méthodes (indicé / pointeur)

IUT Valence
Université Grenoble Alpes

Exercice

Code C

```c
#include <stdio.h>
#include <stdlib.h>

#define TAILLE 10

int * inverser_tableau(int *t, int n) {

    int *resultat = malloc(n*sizeof(int));

    for (int i=0; i<n;i++) {
        resultat[n-i-1] = *(t+i);
    }

    return resultat;
}
```

(à suivre ...)

Exercice

Code C

```c
(...suite)
int main() {

    int t[TAILLE] = {1,2,3,4,5,6,7,8,9,10};

    int *inv = inverser_tableau(t, TAILLE);
    for (int i=0; i<TAILLE;i++) {
        printf("%d ", inv[i]);
    }

    return 0;
}
```

IUT Valence Université Grenoble Alpes

Fin !

IUT Valence
Université Grenoble Alpes