Langage *C* : Pointeurs

Sébastien Jean

IUT de Valence
Département Informatique

v1.0, 13 novembre 2025

IUT Valence
Université Grenoble Alpes

Interlude : Création/clonage d'un projet Gitlab

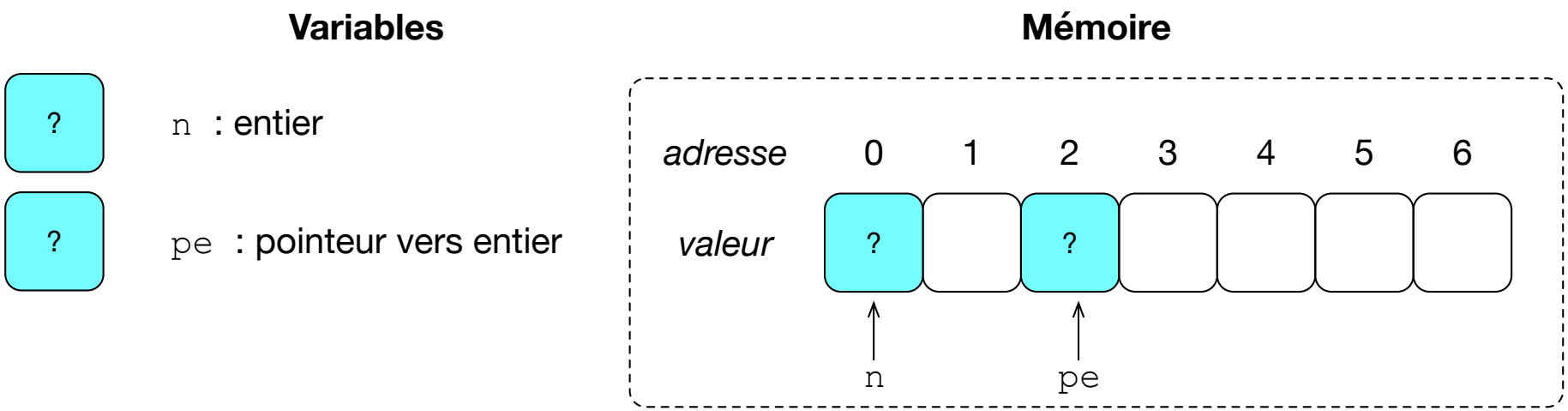
- **Créer un projet** `PointeursC` sur Gitlab (avec un `README`)
- **Cloner** le projet depuis *VsCode* et ouvrir le dépôt
- **Modifier** le fichier `README.md` pour indiquer à quoi sert ce projet
- N.B. : pour chacun des exemples/exercices `ExX` suivant :
  - **Ecrire le programme** dans `ExX/src/main.c`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

IUT Valence Université Grenoble Alpes

Pseudo code : variables (rappels)

- Une **variable de type *pointeur vers un type* `T`** est une **variable dont la valeur est l'adresse d'un emplacement mémoire** où est stockée une valeur de type `T`

```
VARIABLE pe : pointeur vers entier
```



IUT Valence Université Grenoble Alpes

C : variables

- Déclaration d'une variable suivant la syntaxe `type *nom`

Code C

```c
int *p1;          // pointeur sur un entier

double *p2;       // pointeur sur un double

char *p3;         // pointeur sur un caractère
```

- N.B. les **valeurs des pointeurs** en C (les adresses) sont de type `unsigned long`

IUT Valence Université Grenoble Alpes

C : la fonction `sizeof`

- La fonction `sizeof` s'applique à un nom de type ou à une expression
- Elle retourne une valeur de type `unsigned long` qui correspond à l'**espace de stockage occupé** par le type ou l'expression
  - `unsigned long` → `%lu` pour `printf`

Ecrire un programme C qui ...

- Créer une variable de type `bool`, `char`, `short`, `int`, `long`, `float` et `double`
- Créer une variable de type *tableau de 4 cases* pour chacun des types précédents
- Créer une variable de type *pointeur vers* pour chacun des types précédents
- Afficher les espaces de stockage pour chacun des types et chacune des variables

C : la fonction sizeof

Code C

```c
printf("%lu %lu %lu %lu %lu %lu %lu\n", sizeof(bool),
       sizeof(char), sizeof(short), sizeof(int),
       sizeof(long), sizeof(float), sizeof(double));

bool     b;
char     c;
short    s;
int      i;
long     l;
float    f;
double   d;

printf("%lu %lu %lu %lu %lu %lu %lu\n", sizeof(b),
    sizeof(c),
       sizeof(s), sizeof(i), sizeof(l), sizeof(f),
       sizeof(d));
```

C : la fonction sizeof

Code C

```c
bool     tb[4];
char     tc[4];
short    ts[4];
int      ti[4];
long     tl[4];
float    tf[4];
double   td[4];

printf("%lu %lu %lu %lu %lu %lu %lu\n", sizeof(tb),
       sizeof(tc), sizeof(ts), sizeof(ti), sizeof(tl)),
       sizeof(tf), sizeof(td)));

 ...
```

C : la fonction sizeof

Code C

```c
bool     *pb;
char     *pc;
short    *ps;
int      *pi;
long     *pl;
float    *pf;
double   *pd;

printf("%lu %lu %lu %lu %lu %lu %lu\n", sizeof(pb),
       sizeof(pc), sizeof(ps), sizeof(pi), sizeof(pl),
       sizeof(pf), sizeof(pd));
```

IUT Valence
Université Grenoble Alpes

C : taille des types standard (x64)

- `bool` : 1 octet
- `char` : 1 octet
- `short` : 2 octets
- `int` : 4 octets
- `long` : 8 octets
- `float` : 4 octets
- `double` : 8 octets
- `t[x]` : `x` * (taille de `t`) octets
- `t *` : 8 octets

IUT Valence
Université Grenoble Alpes

C : promotion de type

Compiler et exécuter le code C


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

int main() {

  printf("%lu %lu %lu %lu\n", sizeof(true),
         sizeof('c'), sizeof(42), sizeof(1.0));

  printf("%lu %lu %lu %lu\n", sizeof(42L),
         sizeof(1.0f), sizeof(42 + 1L),
         sizeof(1.0f + 1.0));
```

- N.B. : le suffixe `L` impose la valeur a être de type `long`
- N.B. : le suffixe `f` impose la valeur a être de type `float`

IUT Valence Université Grenoble Alpes

C : promotion de type

- Le type standard des entiers est `int`, **toute expression entière de type autre que `int` qui peut être stockée dans un `int`** est **automatiquement promue** en `int`
  - Ex : les valeurs littérales `true`, `'z'` et 42 sont promues vers `int`
- Le type standard des réels est `double`, **toute expression réelle de type `float`** est **promue automatiquement** en `double`

IUT Valence Université Grenoble Alpes

C : promotion de type

- Les opérations arithmétiques **promeuvent automatiquement** les opérandes entières ou réelles pour qu'elles s'adaptent au **type le plus large** (au sens du stockage)
  - `42 + 1L` → `42` promu en `long`, résultat de type `long`
  - `1.0f + 2.0` → `1.0f` promu en `double`, résultat de type `double`
- Lors des appels de fonction, les **paramètres et valeur de retour** sont **promus automatiquement** si besoin
  - *Ex. si la fonction `void add(long a, long b)` est appelée avec des expressions de type `int`, celles-ci sont promues en `long`*

IUT Valence Université Grenoble Alpes

C : conversion de type

- Les opérations arithmétiques **convertissent automatiquement** les opérandes pour qu'elles s'adaptent à la **sémantique des opérations**
  - `42 / 10.3` → `10.3` est un `double`, la division a une sémantique de division réelle, `42` est converti en `double`, le résultat est de type `double`
  - Idem pour `+`, `-` et `*`
- Lors des appels de fonction, les paramètres et valeur de retour sont **convertis automatiquement** si besoin
  - Ex. si la fonction `void add(double a, double b)` est appelée avec des expressions de type `int`, celles-ci sont converties en `double`

IUT Valence Université Grenoble Alpes

C : forçage de type

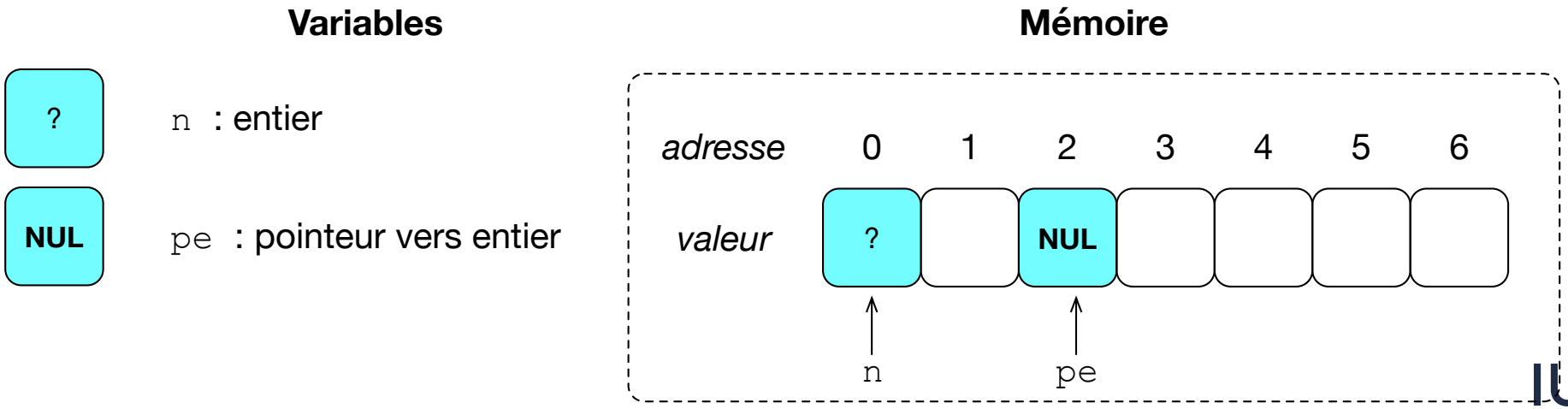
- Le **forçage de type** (ou *cast*) consiste à **imposer un type** à une expression, soit par conversion, soit par réduction de l'espace de stockage
- Le forçage de type peut être automatique
  - `char c = 'z'` → `'z'` de type `int` forcé en `char` (réduction)
  - `int i = 2.0` → `2.0` de type `double` forcé en `int` (conversion)
  - `float f = 2` → `2` de type `int` forcé en `float` (conversion)
- Le forçage de type peut être explicite
  - Le forçage de type s'exprime via la syntaxe `(type) expression`
  - Ex : `(unsigned int) -5` , ou `(double) a / (double) b` force la division à être réelle si `a` et `b` sont des entiers

IUT Valence Université Grenoble Alpes

Pseudo code : Initialisation (rappel)

- Une variable de type *pointeur vers un type T* ne doit pas être lue si elle n'est pas initialisée
- On peut **initialiser une variable de type pointeur** avec la valeur `NUL` pour indiquer qu'elle ne désigne pas encore d'emplacement
  - Elle peut alors être lue, on saura qu'elle ne désigne pas encore d'emplacement

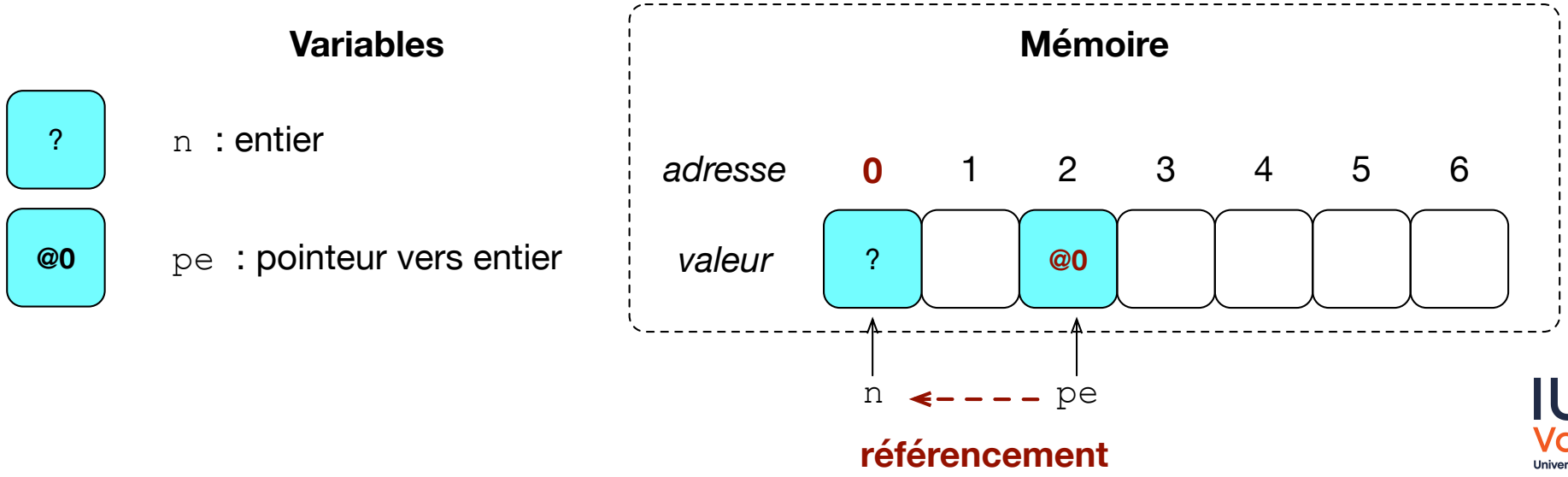
```
pe ← NUL
```



Pseudo code : Initialisation (rappel)

- **On peut affecter comme valeur à une variable de type pointeur l'adresse d'une variable (ou d'un paramètre) du type attendu**
  - L'opérateur & désigne l'**adresse d'une variable**
- La valeur d'une variable de type pointeur vers le type `T` est l'**adresse de l'emplacement mémoire d'une valeur de type `T`**
  - On dit que le pointeur **référence** la variable

```
pe ← &n
```



C : intialisation

Code C

```c
int i = 42;

int *p1;                    // pointeur non initialisé

int *p2 = NULL;             // pointeur initialisé
                            // à nul

int *p3 = &i;               // pointeur initialisé
```

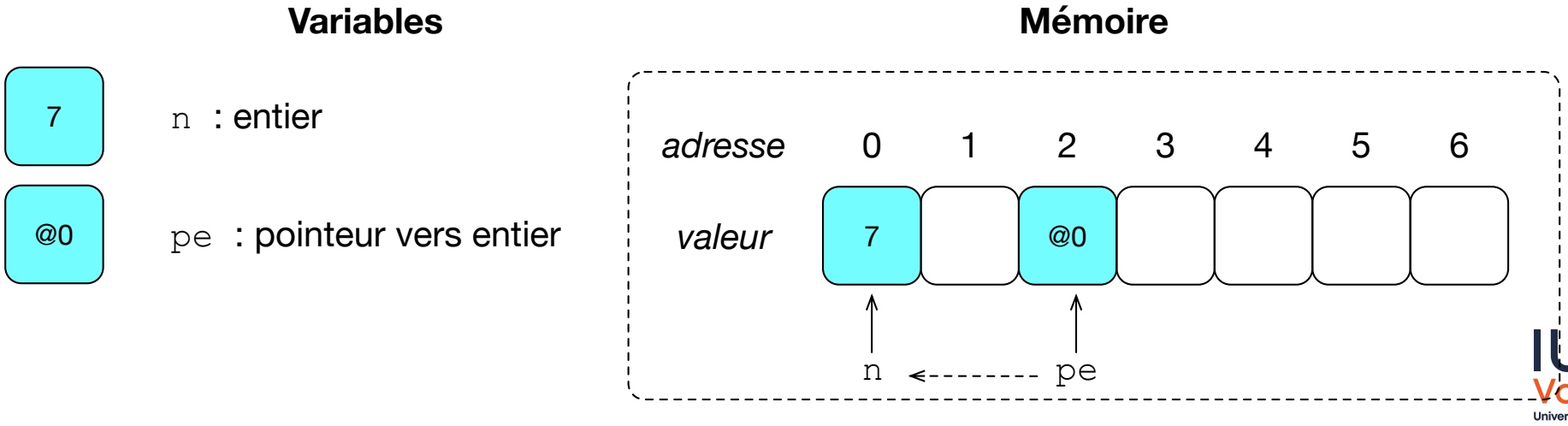
- Afficher p2 et p3.
  - N.B. : %p pour le formattage de pointeurs dans printf

IUT Valence Université Grenoble Alpes

Pseudo Code : déréférencement de pointeur (rappel)

- L'opérateur ↑ exprime le **déréférencement**, c'est à dire la **désignation de l'emplacement mémoire dont l'adresse est contenue dans le pointeur**
- L'opérateur ↑ peut être utilisé dans une expression
  - Dans ce cas l'**expression vaut la valeur contenue dans l'emplacement dont l'adresse est contenue dans le pointeur**

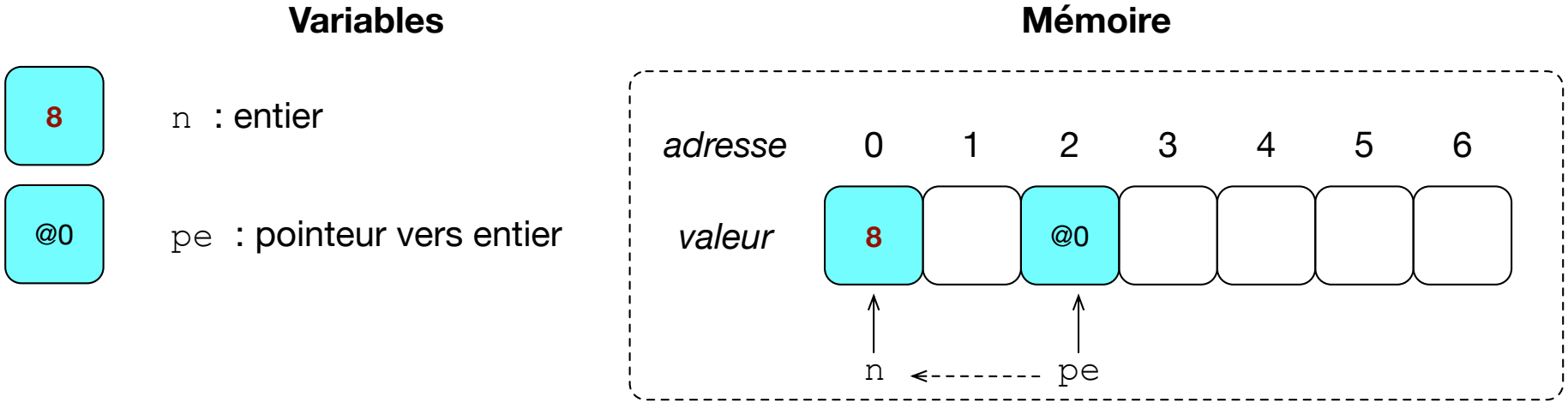
```
afficher_entier(pe↑)                    // affiche 7
```



Pseudo Code : déréférencement de pointeur (rappel)

- L'opérateur ↑ peut être utilisé dans une affectation
  - Dans ce cas l'**emplacement dont l'adresse est contenue dans le pointeur est réaffecté avec la nouvelle valeur**

```
pe↑ ← 8
```



C : déréférencement de pointeur

- Le **déréférencement de pointeur** s'effectue via la syntaxe `*pointeur`
  - Dans une expression → valeur contenue à l'@ désignée par le pointeur
  - Dans une affectation → réaffectation de la valeur contenue à l'adresse désignée par le pointeur

Code C

```c
int i = 42;
int *p = &i;
printf("%p %d\n", p, *p);

*p = 1;
printf("%p %d\n", p, *p);
```

Exercice (version naïve)

Enoncé du problème

On veut permuter les valeurs de 2 entiers.

Spécification du problème

- **Donnée d'entrée** : *e1*, entier (premier)
- **Donnée d'entrée** : *e2*, entier (second)
- **Donnée de sortie** : (aucune)
- **Pré-condition** : (aucune)
- **Post-condition** : les 2 valeurs sont permutées.

Signature de la fonction

- **permute** (e1 : entier, e2 : entier) : (aucun)

IUT Valence
Université Grenoble Alpes

Exercice (version correcte)

Enoncé du problème

On veut permuter les valeurs de 2 entiers.

Spécification du problème

- **Donnée d'entrée** : *pe1*, pointeur vers un entier (premier)
- **Donnée d'entrée** : *pe2*, pointeur vers un entier (second)
- **Donnée de sortie** : (aucune)
- **Pré-condition** : $pe1 \neq NUL$ ET $pe2 \neq NUL$
- **Post-condition** : les 2 valeurs sont permutées.

Signature de la fonction

- **permute** (pe1 : pointeur vers entier, pe2 : pointeur vers entier) : (aucun)

IUT Valence
Université Grenoble Alpes

Fin !

IUT Valence
Université Grenoble Alpes