

Langage C

Les pointeurs



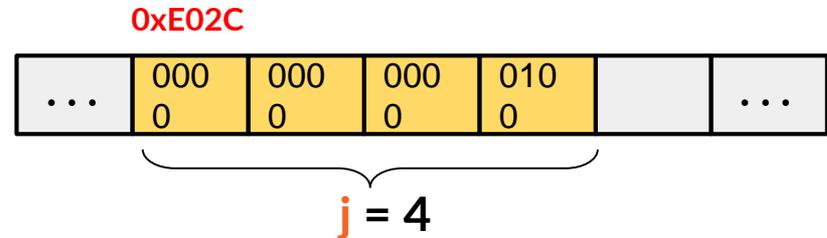
`#include<stdio.h>`

Variables et adresses mémoire

Déclaration Initialisation

`int j = 4;` ← `int` codé sur 4 octets

Mémoire vive



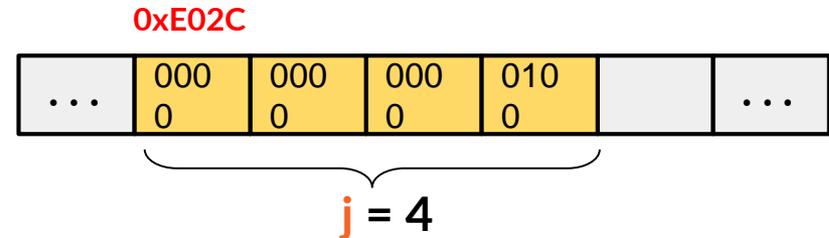
1. La mémoire est allouée
2. Le nom de la variable `j` est lié à l'adresse de la case `0xE02C`
3. la valeur 4 est écrite en binaire dans l'espace mémoire

Variables et adresses mémoire

Déclaration Initialisation

`int j = 4;` ← `int` codé sur 4 octets

Mémoire vive



1. La mémoire est allouée
2. Le nom de la variable `j` est lié à l'adresse de la case `0xE02C`
3. la valeur 4 est écrite en binaire dans l'espace mémoire

- **Peut on accéder à l'adresse d'une variable ?** Oui ! Avec l'opérateur `&`
`&j` représente l'adresse de la variable `j`, soit `0xE02C`
- **Peut on manipuler une variable par son adresse ?** Oui ! Avec les **pointeurs**

Qu'est ce qu'un pointeur ?

Pointeur : variable qui peut stocker l'adresse d'une autre variable

```
int j = 4;
```

```
int* p_j; ←
```

Déclaration

p_j va stocker l'adresse (de la 1^{ère} case) d'un **int**

Qu'est ce qu'un pointeur ?

Pointeur : variable qui peut stocker l'adresse d'une autre variable

```
int j = 4;
```

```
int* p_j; ←
```

Déclaration

p_j va stocker l'adresse (de la 1^{ère} case) d'un **int**

```
p_j = &j; ←
```

Initialisation

On stocke l'adresse de j dans p_j (on dit que “p_j pointe sur j”)

```
*p_j = 2;  
par p_j
```

L'opérateur * permet d'accéder au contenu de la variable pointée

Qu'est ce qu'un pointeur ?

Pointeur : variable qui peut stocker l'adresse d'une autre variable

```
int j = 4;
```

```
int* p_j; ←
```

Déclaration

`p_j` va stocker l'adresse (de la 1^{ère} case) d'un `int`

```
p_j = &j; ←
```

Initialisation

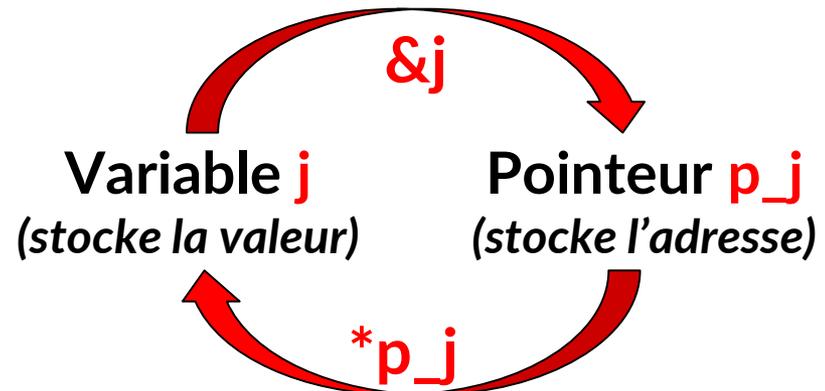
On stocke l'adresse de `j` dans `p_j` (on dit que “`p_j` pointe sur `j`”)

```
*p_j = 2;  
par p_j
```

L'opérateur `*` permet d'accéder au contenu de la variable pointée

Bonne pratique:

Un pointeur sera nommé `p_X`



Création d'un pointeur

```
int j = 4;
```



```
int* p_j;
```

```
p_j = &j;
```

```
*p_j = 2;
```

Mémoire vive



0xE02C

j = 4

Allocation d'un espace mémoire pour stocker
l'int j et écriture de la valeur 4



Création d'un pointeur

```
int j = 4;
```

```
int* p_j;
```



```
p_j = &j;
```

```
*p_j = 2;
```

Mémoire vive



0xE02C

j = 4

Allocation d'un espace mémoire pour stocker
l'adresse d'un `int`



p_j

Création d'un pointeur

```
int j = 4;
```

```
int* p_j;
```

```
p_j = &j;
```

```
*p_j = 2;
```

Mémoire vive



0xE02C

j = 4

On donne la valeur &j (l'adresse de j)
à la variable p_j



p_j

Création d'un pointeur

```
int j = 4;
```

```
int* p_j;
```

```
p_j = &j;
```

```
*p_j = 2; ←
```

Mémoire vive



0xE02C

j = 2



p_j

On donne la valeur 2 à l'entier *p_j
qui est stocké à l'adresse p_j

Création d'un pointeur

```
int j = 4;  
int* p_j;
```

```
p_j = &j;
```

```
*p_j = 2;
```

```
printf("j = %d", j);
```

← **j = 2**

On a manipulé l'**int** j à travers son adresse mémoire

Mémoire vive



0xE02C

j = 2



p_j

Bonne initialisation d'un pointeur

```
int* p;
```

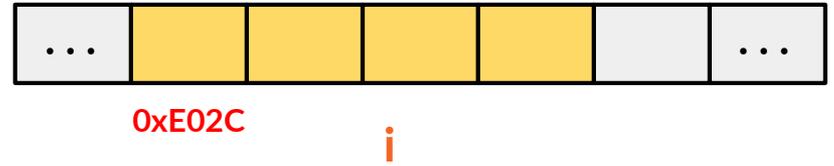
```
int i;
```

```
p = &i;
```

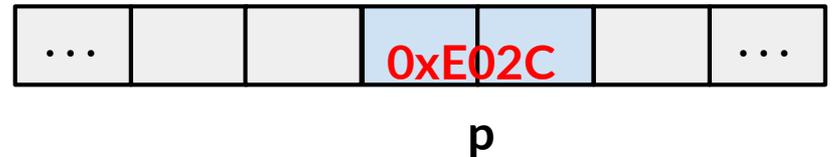
```
*p = 2;
```



Mémoire vive



1. Allocation d'un espace mémoire pour stocker `i`
2. On initialise `p` avec l'adresse de `i`



Mauvaise initialisation d'un pointeur

```
int* p;
```



Mémoire vive



Allocation d'un espace mémoire pour stocker
l'adresse d'un `int`



```
*p = 2;
```

Mauvaise initialisation d'un pointeur

```
int* p;
```



Pas d'erreur à la compilation mais
plantage à l'exécution

```
*p = 2;
```



Mémoire vive



On écrit la valeur 2 dans la case pointée par p

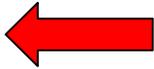


Problème:
p ne pointe vers aucune case

Pourquoi s'embêter avec des pointeurs ?

Retour sur les fonctions...

```
int j = 4;
```



```
...
```

```
Foisdeux(j);
```

```
printf("%d", j);
```

```
void Foisdeux(int j){
```

```
    j = j*2;
```

```
}
```

Mémoire vive



0xE02C

j = 4



Pourquoi s'embêter avec des pointeurs ?

Retour sur les fonctions...

```
int j = 4;
```

```
...
```

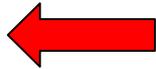
```
Foisdeux(j);
```

```
printf("%d", j);
```

```
void Foisdeux(int j){
```

```
    j = j*2;
```

```
}
```



**j est une
variable locale**

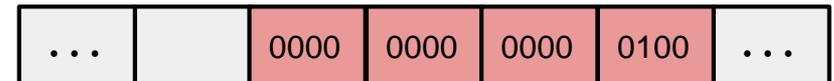
Mémoire vive



0xE02C

j = 4

copie de j = 4



0xF09A

Pourquoi s'embêter avec des pointeurs ?

Retour sur les fonctions...

```
int j = 4;
```

```
...
```

```
Foisdeux(j);
```

```
printf("%d", j);
```

```
void Foisdeux(int j){
```

```
    j = j*2;
```

```
}
```



**j est une
variable locale**

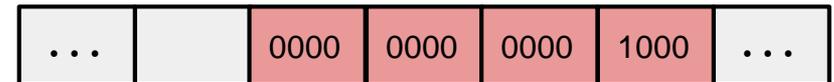
Mémoire vive



0xE02C

j = 4

copie de j = 8



0xF09A

Pourquoi s'embêter avec des pointeurs ?

Retour sur les fonctions...

```
int j = 4;
```

```
...
```

```
Foisdeux(j);
```

```
printf("%d", j);
```



Qu'affiche
printf ?



j est une
variable locale

```
void Foisdeux(int j){
```

```
    j = j*2;
```

```
}
```

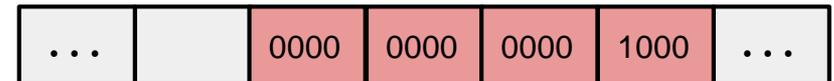
Mémoire vive



0xE02C

j = 4

copie de j = 8



0xF09A

Pourquoi s'embêter avec des pointeurs ?

Retour sur les fonctions...

```
int j = 4;
```

```
...
```

```
Foisdeux(j);
```

```
printf("%d", j);
```

 **j = 4**

```
void Foisdeux(int j){
```

```
    j = j*2;
```

```
}
```



**j est une
variable locale**

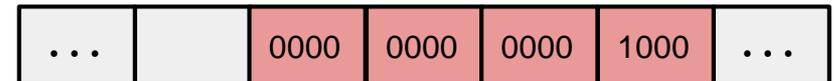
Mémoire vive



0xE02C

j = 4

copie de j = 8

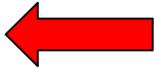


0xF09A

Pourquoi s'embêter avec des pointeurs ?

Retour sur les fonctions...

```
int j = 4;
```



```
...
```

```
Foisdeux(&j);
```

```
printf("%d", j);
```

Pointeur:

Passage d'arguments

par adresse

```
void Foisdeux(int* p){
```

```
    (*p) = (*p)*2;
```

```
}
```

Mémoire vive



0xE02C

j = 4



Pourquoi s'embêter avec des pointeurs ?

Retour sur les fonctions...

```
int j = 4;
```

```
...
```

```
Foisdeux(&j);
```

```
printf("%d", j);
```

```
void Foisdeux(int* p){
```

```
    (*p) = (*p)*2;
```

```
}
```

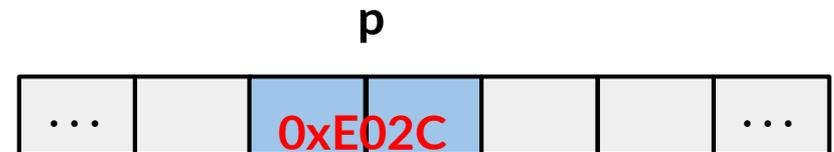
Pointeur:
Passage d'arguments
par adresse

Mémoire vive



0xE02C

j = 4



p

Allocation d'un espace mémoire pour stocker l'adresse d'un `int` et écriture de l'adresse de `j`

Pourquoi s'embêter avec des pointeurs ?

Retour sur les fonctions...

```
int j = 4;
```

```
...
```

```
Foisdeux(&j);
```

```
printf("%d", j);
```

Pointeur:

Passage d'arguments

par adresse

```
void Foisdeux(int* p){
```

```
    (*p) = (*p)*2;
```

```
}
```

Mémoire vive



0xE02C

j = 8

p



On multiplie par 2 l'entier *p stocké à l'adresse p

Pourquoi s'embêter avec des pointeurs ?

Retour sur les fonctions...

```
int j = 4;
```

```
...
```

```
Foisdeux(&j);
```

```
printf("%d", j); ← j = 8
```

Plus besoin du return...

```
void Foisdeux(int* p){
```

```
    (*p) = (*p)*2;
```

```
}
```

Mémoire vive



0xE02C

j = 8

p



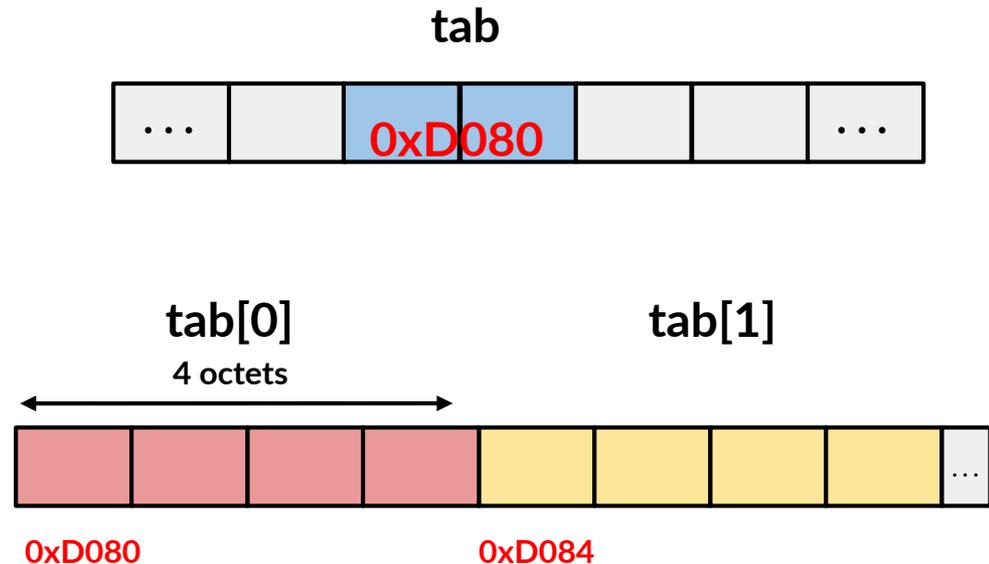
Pourquoi s'embêter avec des pointeurs ?

*Un tableau est un **pointeur** + un **bloc mémoire***

```
int tab[5];
```

```
tab[0] = 1;
```

```
*(tab+1) = 4;
```



1. Définition d'un pointeur `tab`
2. Allocation d'un bloc mémoire de 5 x 4 octets contigus

tab contient l'adresse du 1er élément du tableau

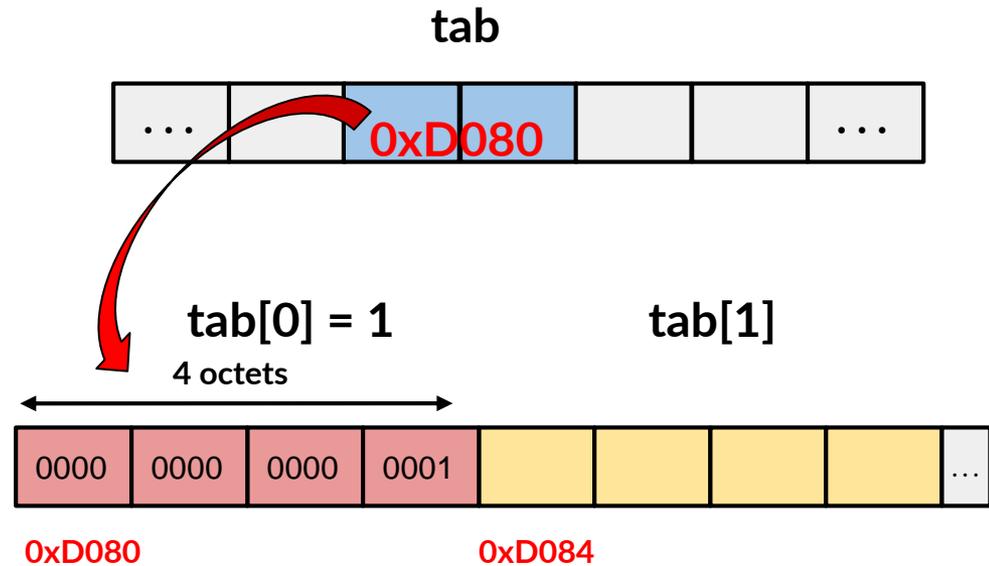
Pourquoi s'embêter avec des pointeurs ?

*Un tableau est un **pointeur** + un **bloc mémoire***

```
int tab[5];
```

```
tab[0] = 1; ←
```

```
*(tab+1) = 4;
```



On écrit la valeur 1 dans la 1ere case

Pourquoi s'embêter avec des pointeurs ?

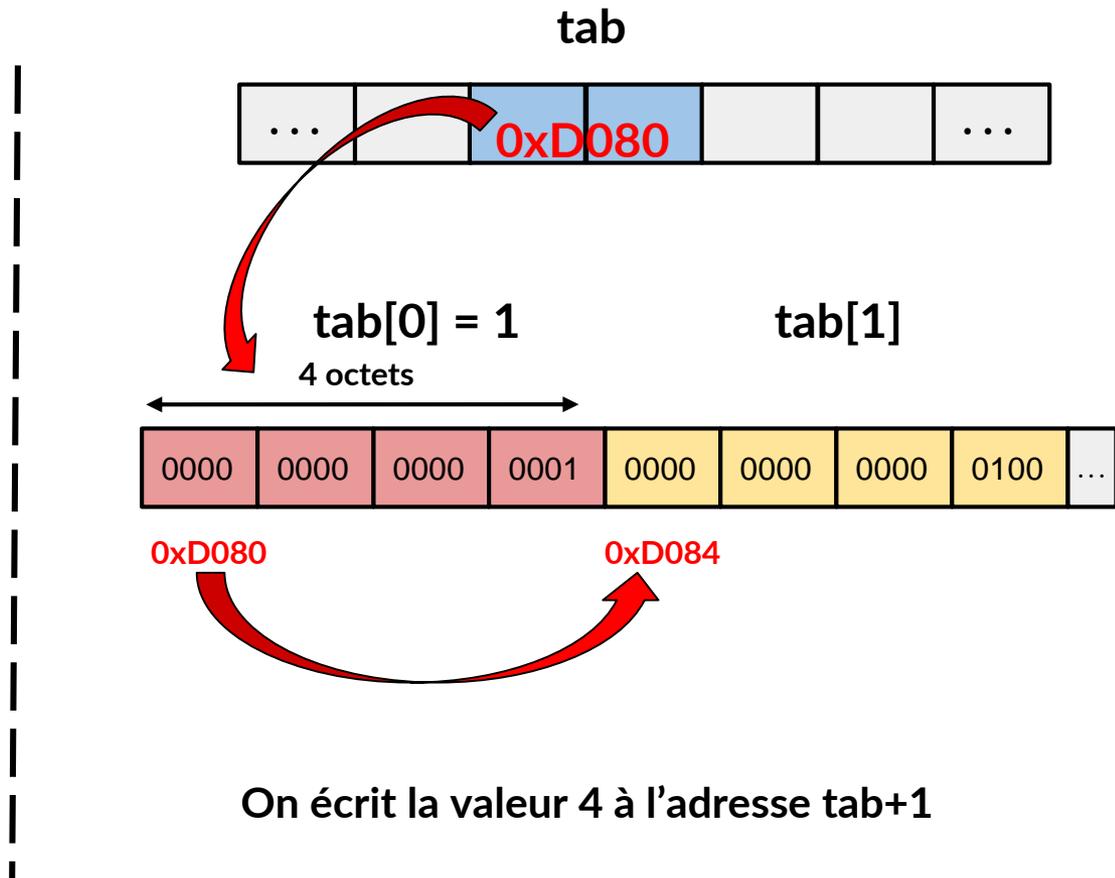
Un tableau est un **pointeur** + un **bloc mémoire**

```
int tab[5];
```

```
tab[0] = 1;
```

```
*(tab+1) = 4;
```

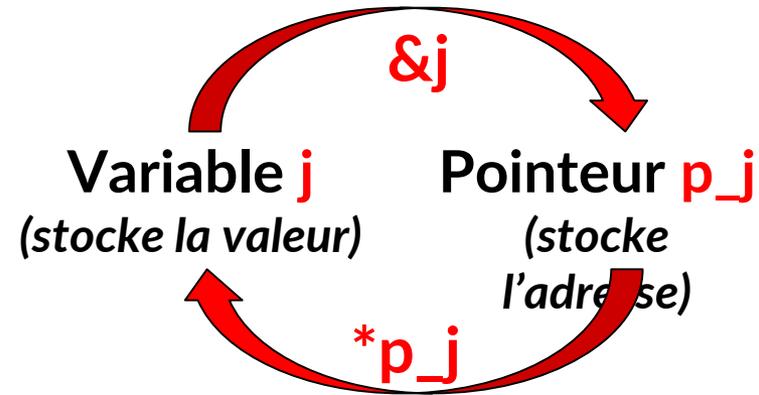
tab+1 pointe sur l'entier suivant en mémoire



Bilan sur les pointeurs

Pointeur : variable qui peut stocker l'adresse d'une autre variable

- Permet de manipuler des variables par leur adresse (unique dans la mémoire)
- Passage d'arguments par adresse



`void Mafonction(int* p1, int* p2, ...);`

=> **Nombre illimité d'arguments de sortie**

- Tableau = pointeur + bloc mémoire

`tab[4] ⇔ *(tab + 4)`

`&tab[2] ⇔ tab + 2`