# Les pointeurs en C

Séance 5

de l'UE « introduction à la programmation »

Bruno Bouzy

bruno.bouzy@parisdescartes.fr

# Pointeurs

- Définition, utilisation de base

- Utilisation avec les fonctions

- Utilisation avec les tableaux

- Utilisation avec les structures

# Déclaration

- **Déclaration** d'un pointeur sur `int`

```
int * p;
```

# Initialisation

- Convention d'**initialisation** avec `NULL`

```
p = NULL;
```

# Affectation

- **Affectation avec une adresse** de variable

```
int a;
...
p = &a;
```

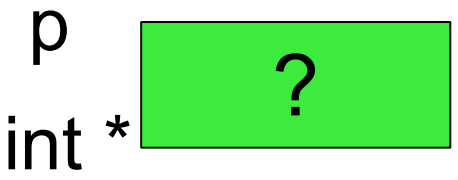
# Déréférencement

- **Déréférencement** d'un pointeur

```
int b;
b = *p;
```

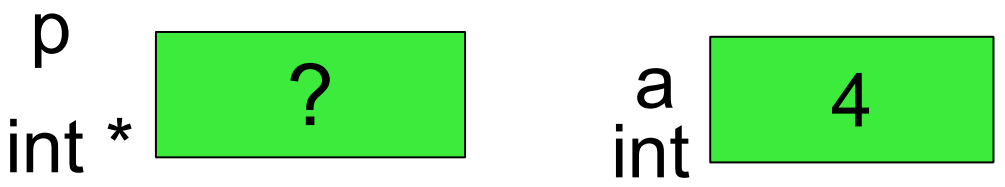
# Exemple basique (1/5)

```
int * p;
```



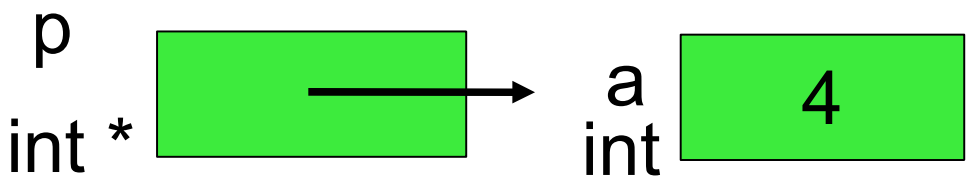
# Exemple basique (2/5)

```
int * p;
int a=4;
```



# Exemple basique (3/5)

```
int * p;
int a=4;
p = &a;
```



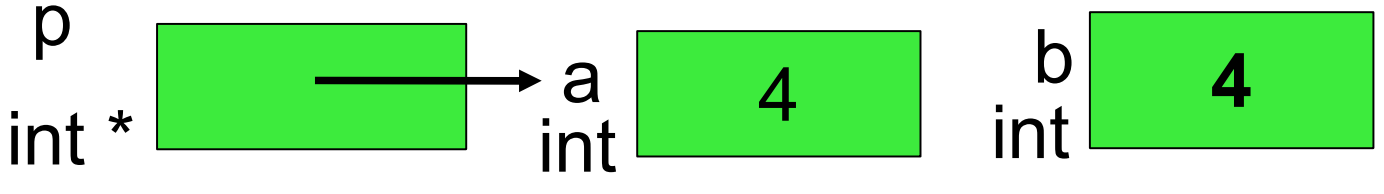

- `p` « pointe » sur `a`
- `&a` : adresse de `a`

# Exemple basique (4/5)

```
int * p;
int a=4;
p = &a;
int b;
```

# Exemple basique (5/6)

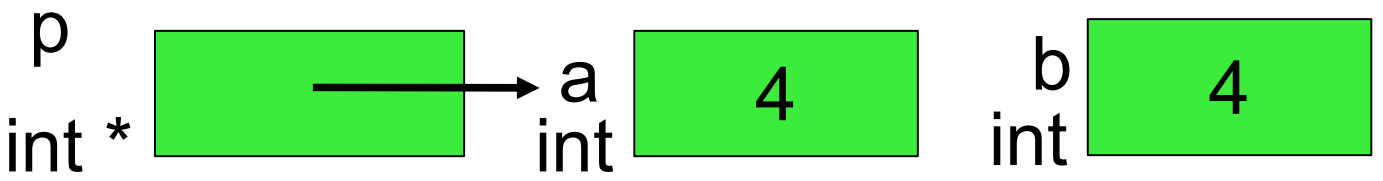
```
int * p;
int a=4;
p = &a;
int b;
b = *p;
```




- `*p` : contenu de la variable « pointée » par `p`
- `*p` : « déréférencement » de `p`

# Exemple basique (6/6)

```
int a=4; int * p = &a;
int b; b = *p;
```




- `p` « pointe » sur `a`
- `&a` adresse de `a`
- `*p` contenu de la variable pointée par `p` ou « déréférencement » de `p`

# Type d'un pointeur

```
float a=4.0; int * p = &a;
```

- Erreur de compilation: un pointeur est un pointeur sur des variables d'un type donné.

```
float a=4.0; float * p = &a;
```

- Une variable de type `truc *` pointe sur un `truc`

# Les erreurs

- Un pointeur non initialisé sur une adresse de variable ne peut etre déréférencé.

```
int * p; int b; b = *p;
```

- Erreur d'exécution.

```
int * p = NULL; int b; b = *p;
```

- Erreur d'exécution.

# Utilisation de `NULL` (1/3)

- Pourquoi initialiser un pointeur avec `NULL` ?
- 3 cas: le pointeur pointe sur
  - la variable adéquate et on peut s'en servir
  - une variable inadéquate et le programme est incorrect
  - `NULL`
- Convention:
  - On affecte `NULL` quand un pointeur ne sert plus ou pas encore

# Utilisation de `NULL` (2/3)

- Avec cette convention, 2 cas:
  - Pointeur sur variable adéquate
  - Pointeur `NULL`

- Avant d'utiliser un pointeur on teste sa valeur:

```
if (p!=NULL) { ... b = *p; ...}
else { ... }
```

# Utilisation de `NULL` (3/3)

- Avec cette convention le pointeur est mis à `NULL`
  - Après utilisation:

```
p = &a; ... b = *p; ... p = NULL;
```

  - A la déclaration (si non simultanée de l'affectation):

```
int * p = NULL;    ... p = &a;
```

# Exemple 1

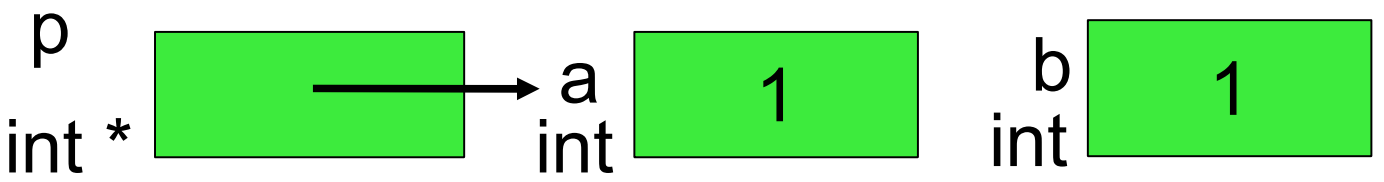
- `printf("a = %d, b = %d, *p = %d.\n", a, b, *p);` après chaque ligne.

- sortie du programme ?

```
int a = 1; int * p = &a; int b = *p;
a = 2;
b = 3;
p = &b;
a = 4;
b = 5;
```

# Exemple 1

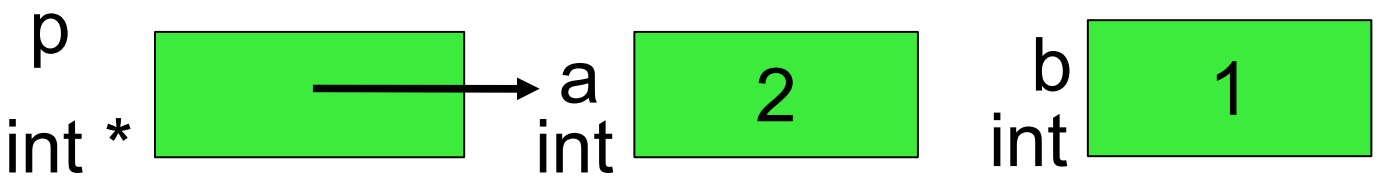
```
int a = 1; int * p = &a; int b = *p;
```




```
a = 1, b = 1, *p = 1.
```

# Exemple 1

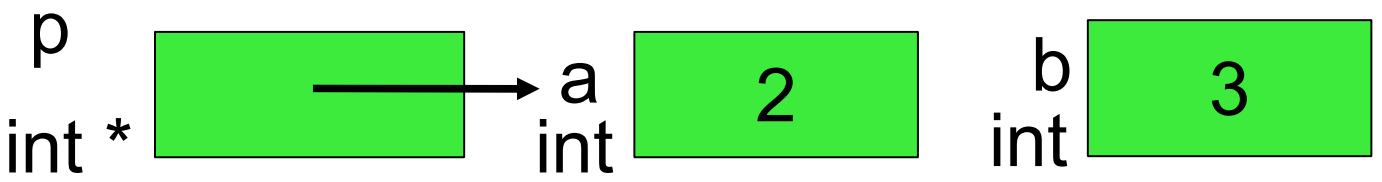
```
int a = 1; int * p = &a; int b = *p;
a = 2;
```




a = **2**, b = 1, *p = **2**.

# Exemple 1

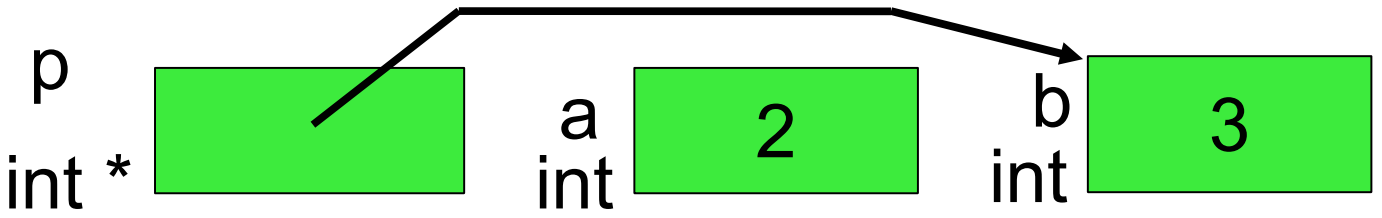
```
int a = 1; int * p = &a; int b = *p;
a = 2;
b = 3;
```




`a = 2, b = 3, *p = 2.`

# Exemple 1

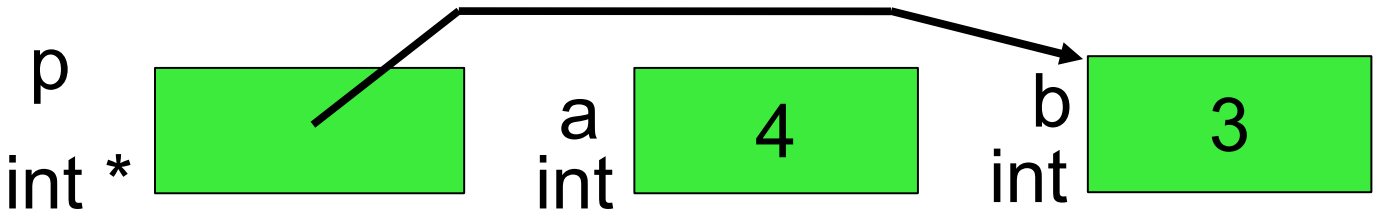
```
int a = 1; int * p = &a; int b = *p;
a = 2;
b = 3;
p = &b;
```




`a = 2, b = 3, *p =` **`3.`**

# Exemple 1

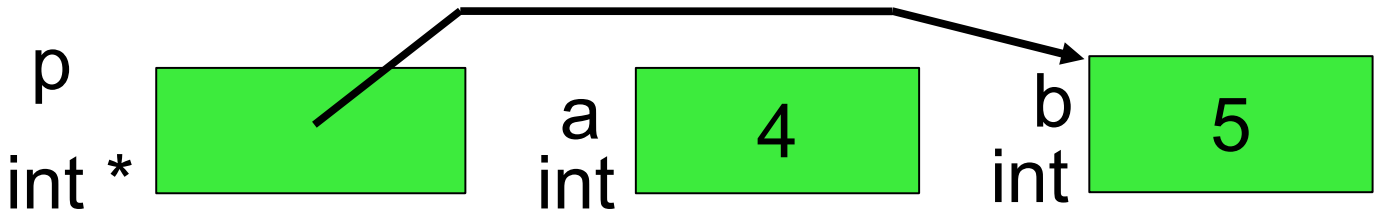
```
int a = 1; int * p = &a; int b = *p;
a = 2;
b = 3;
p = &b;
a = 4;
```




a = **4**, b = 3, *p = 3.

# Exemple 1

```
int a = 1; int * p = &a; int b = *p;
a = 2;
b = 3;
p = &b;
a = 4;
b = 5;
```




a = 4, b = **5**, *p = **5**.

# Exemple 1

- sortie:

```
a = 1, b = 1, *p = 1.
a = 2, b = 1, *p = 2.
a = 2, b = 3, *p = 2.
a = 2, b = 3, *p = 3.
a = 4, b = 3, *p = 3.
a = 4, b = 5, *p = 5.
```

- 2 changements d'un coup si on modifie le contenu d'une variable pointée.

# Exemple 2

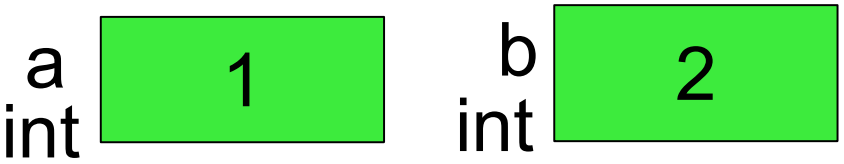
`printf("a = %d, b = %d, *p = %d, *q = %d.\n", a, b, *p, *q);` après chaque ligne (sauf ligne 1 et 2).

- sortie du programme ?

```
int a = 1; int b = 2;
int * p = &a;
int * q = p;
a = 3;
b = *q + 10;
```

# Exemple 2

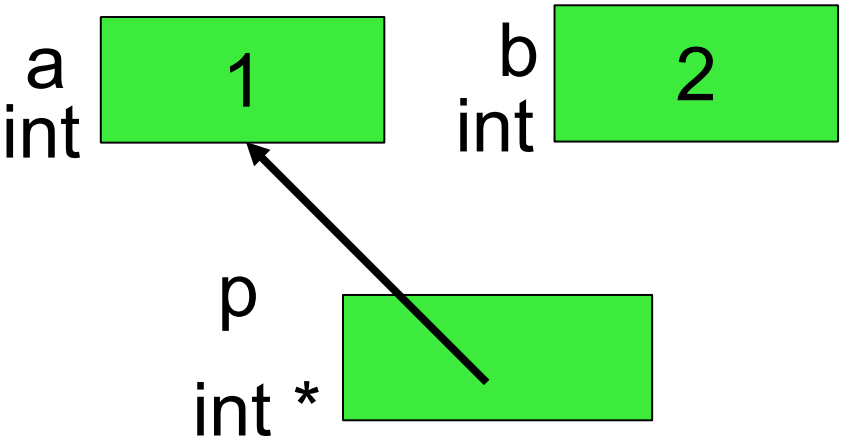
```
int a = 1; int b = 2;
```




a = **1**, b = **2**.

# Exemple 2

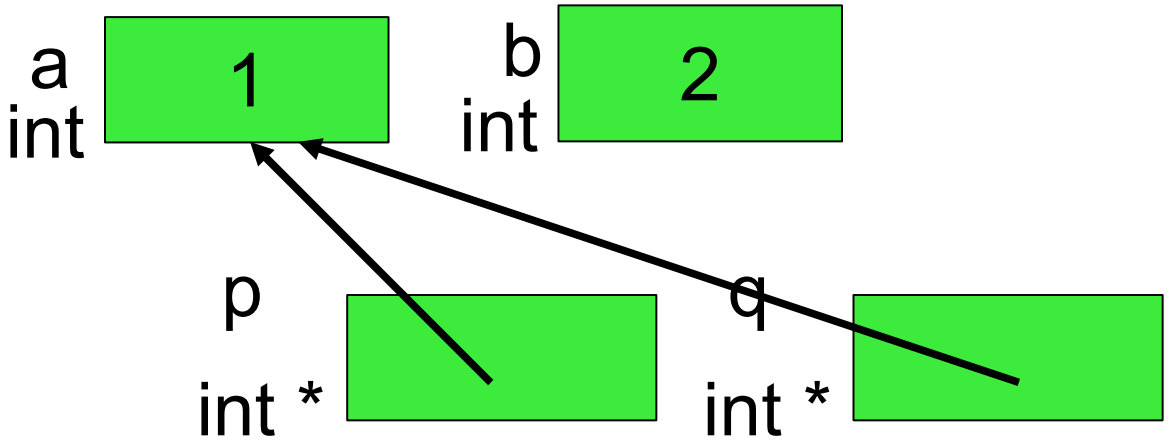
```
int a = 1; int b = 2;
int * p = &a;
```




```
a = 1, b = 2, *p = 1
```

# Exemple 2

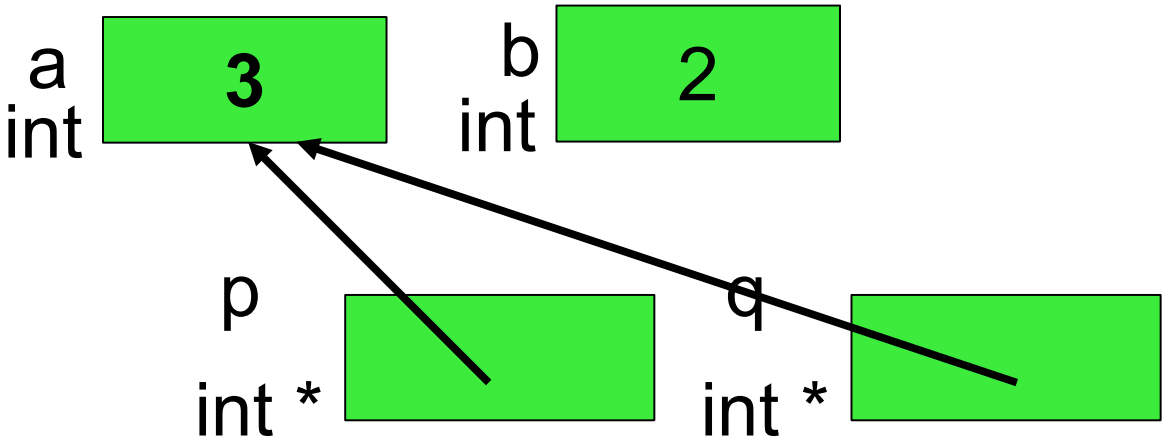
```
int a = 1; int b = 2;
int * p = &a;
int * q = p;
```




```
a = 1, b = 2, *p = 1, *q = 1.
```

# Exemple 2

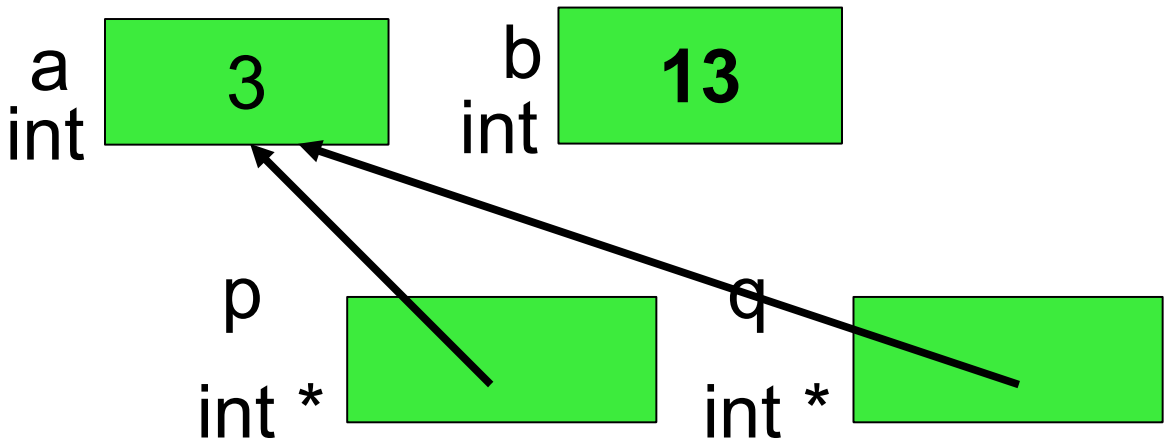
```
int a = 1; int b = 2;
int * p = &a;
int * q = p;
a = 3;
```




a = **3**, b = 2, *p = **3**, *q = **3**.

# Exemple 2

```
int a = 1; int b = 2;
int * p = &a;
int * q = p;
a = 3;
b = *q + 10;
```




a = 3, b = **13**, *p = 3, *q = 3.

# Exemple 2

```
a = 1, b = 2, *p = 1, *q = 1.
a = 3, b = 2, *p = 3, *q = 3.
a = 3, b = 13, *p = 3, *q = 3.
```

- 3 changements en une instruction.

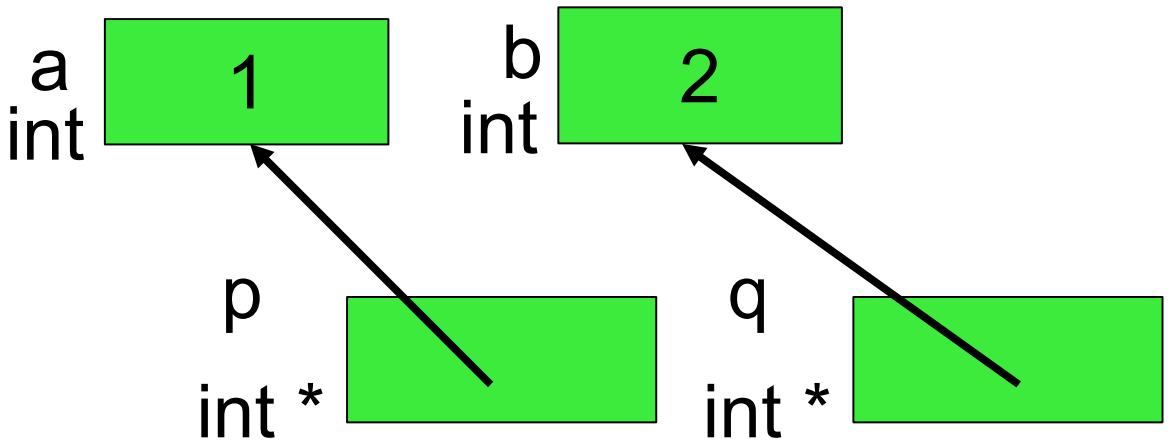
# Exemple 3

- Sortie ?

```
int a = 1; int b = 2;
int * p = &a;  int * q = &b;
printf("a  = %d, b  = %d.\n", a,   b);
printf("*p = %d, *q = %d.\n", *p, *q);
printf("&a = %p, &b = %p.\n", &a, &b);
printf(" p = %p,  q = %p.\n", p,   q);
```

# Exemple 3

```
int a = 1; int b = 2;
int * p = &a;  int * q = &b;
```



# Exemple 3

```
a  = 1, b  = 2.
*p = 1, *q = 2.
&a = 0xbffcfab4, &b = 0xbffcfab0.
 p = 0xbffcfab4,  q = 0xbffcfab0.
```

- Ici, les adresses de `a` et `b` se suivent.
  - `&b = &a - 4` (en octets)
  - (non garanti)
- `%p` est le format d'entrée sortie d'un pointeur.

# Résumé de la séance 5

- Pointeurs, utilisation:
  - basique
  - (avec des fonctions
  - avec des tableaux
  - avec des structures)

- Exemples