UNIVERSITÉ PARIS 8
VINCENNES-SAINT-DENIS

Licence informatique & vidéoludisme
Semestre 1

Méthodologie de la programmation

Chapitre 3

La programmation en C (partie 2)

1. Passage de paramètres

2. Adresses et données

3. Pointeurs

Différents modes de passages des paramètres

- Passage par valeur : la **valeur** de l'expression passée en paramètre est copiée dans une variable locale.
- Passage par variable : la **variable** elle-même est passée en paramètre.
  conséquence :
  *Toute **modification du paramètre** dans la fonction appelée entraîne la **modification de la variable** passée en paramètre*.

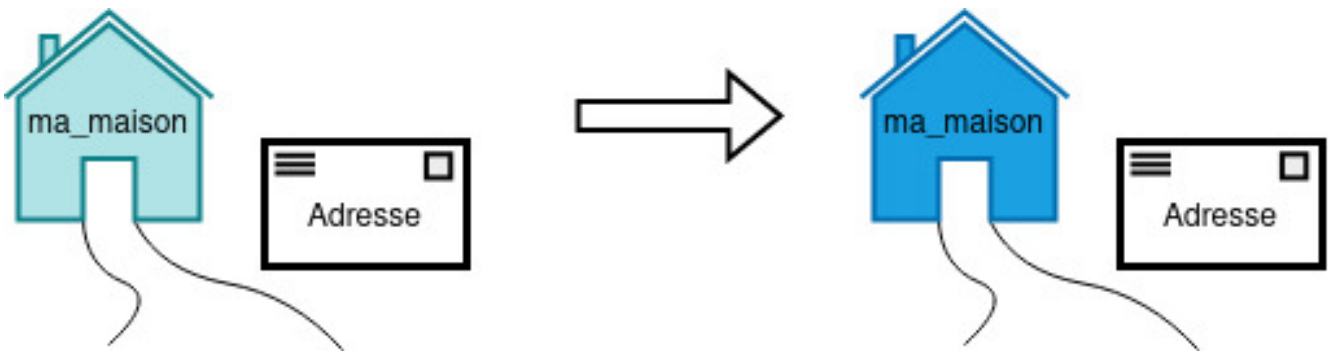
Différents modes de passages des paramètres

- Passage par valeur : la **valeur** de l'expression passée en paramètre est copiée dans une variable locale.
- Passage par variable : la **variable** elle-même est passée en paramètre.
  conséquence :
  *Toute **modification du paramètre** dans la fonction appelée entraîne la **modification de la variable** passée en paramètre*.

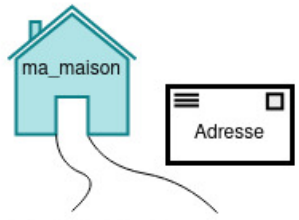
C n'autorise **pas** le passage par variable

- Passage par référence : l'**adresse** de la variable est passée en paramètre.

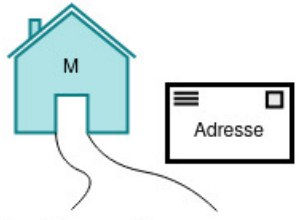
Exemple : repeindre une maison



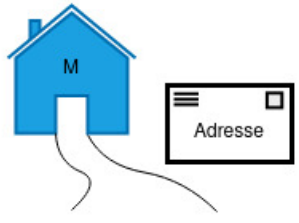
Passage de paramètre par VARIABLE



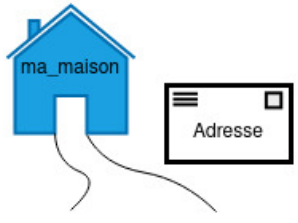

```
def peindre(M, bleu) :
```



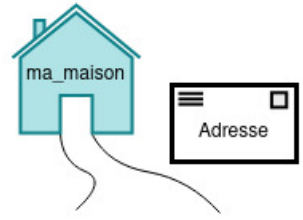

```
M.couleur = bleu
```



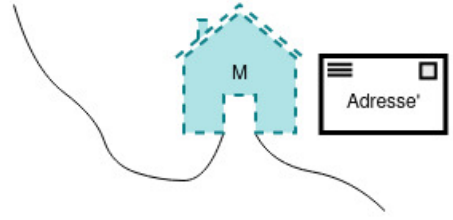

```
peindre(ma_maison, bleu)
```



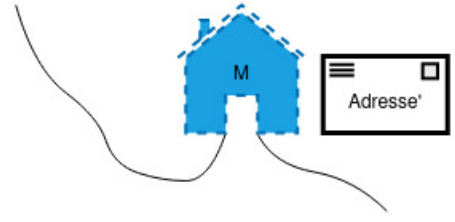
Passage de paramètre par VALEUR



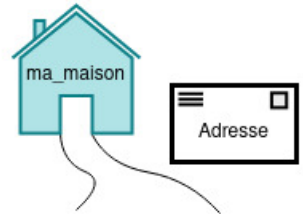

```
void peindre(Maison M, Couleur bleu) {
```




```
M.couleur = bleu
```

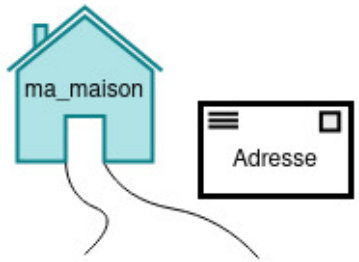



```
}
peindre(ma_maison, bleu) ;
```

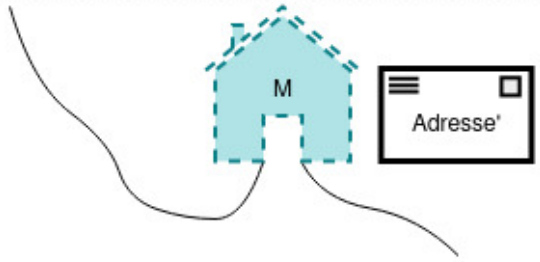


Passage de paramètre par VALEUR

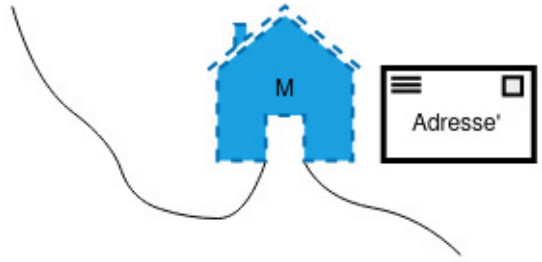
une solution



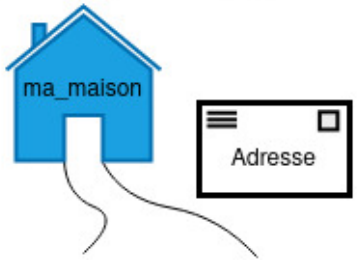

```
Maison peindre(Maison M, Couleur bleu) {
```




```
M.couleur = bleu;
```




```
return M ;
}
ma_maison = peindre(ma_maison, bleu)
```

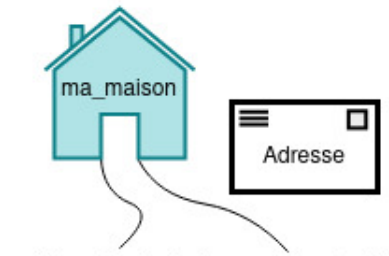


Passage par référence

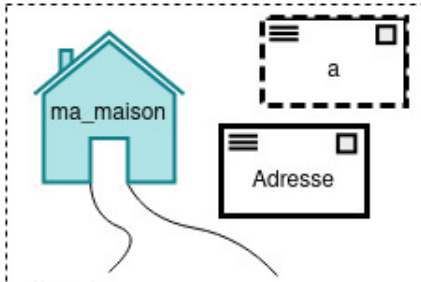
référence : Action de (se) référer à quelqu'un, à quelque chose, de le/la désigner.

- mon amie Émilie Paillous
- la page 43 du manuel
- la maison du 7, rue des Chataîgners
- l'ordinateur N° 451475AZ7

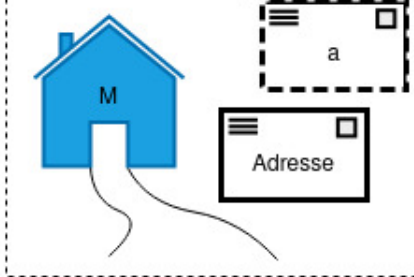
Passage de paramètre par REFERENCE



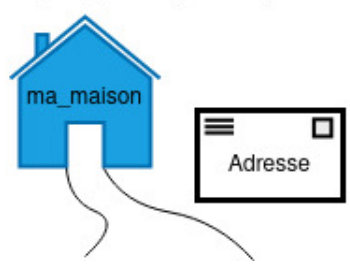

```
void peindre(adresse a, Couleur bleu) {
```




```
M = *a
M.couleur = bleu;
```




```
}
peindre(&ma_maison, bleu)
```



Autres exemples ?

1. Passage de paramètres

2. Adresses et données

3. Pointeurs

Vocabulaire à maîtriser

- la mémoire est une suite d'**octets**
- chaque **octet** a une adresse donnée par une suite de ? bits
- si l'adresse d'un octet est x, l'adresse de l'octet suivant est x+1
- les données sont stockées dans des octets (ou groupes d'octets contigus)

fichier `test_adresses.c`

Vocabulaire à maîtriser

- la mémoire est une suite d'**octets**
- chaque **octet** a une adresse donnée par une suite de ? bits
- si l'adresse d'un octet est x, l'adresse de l'octet suivant est x+1
- les données sont stockées dans des octets (ou groupes d'octets contigus)

fichier `test_adresses.c`

1. Combien de bits composent l'adresse d'un objet de type `char` ? d'un objet de type `int` (32 bits) ?
2. Sur combien d'octet(s) est stocké un objet de type `char` ? un objet de type `int` (32 bits) ?

fichier `exemple_tailles.c`

1. Passage de paramètres

2. Adresses et données

3. Pointeurs

Pointeurs : définition

Un objet pointeur est défini par :

- une **adresse**
- un **type** : nécessaire pour indiquer combien d'octets lire lorsqu'on souhaite manipuler l'objet pointé

```
1 type *p; // déclaration d'un pointeur p
```

une fois p défini :

- l'opérateur $*$ permet d'accéder à la valeur pointée, c'est le déréférencement ;
- l'opérateur & permet de récupérer l'adresse mémoire d'une variable ;
- le format **%p** de `printf` permet d'afficher une adresse mémoire en hexadécimal.

Pointeur sans type

on peut définir un pointeur sans type :

```
1 void  *ptr;
```

Dans ce cas, `ptr` ne sert qu'à stocker une adresse mémoire, mais on ne pourra pas utiliser l'opérateur $*$, pourquoi ?

Pointeurs : bonne pratique

Il est recommandé de toujours initialiser un pointeur, même à NULL, afin de pouvoir tester sa valeur avant de l'utiliser :

```
1 int32_t i = -12;
2 int32_t *p_init = &i;    /* pointe sur l'adresse de i */
3 int32_t *p_ok = NULL;    /* vaut 0x0 */
4 int32_t *p_danger;       /* a une valeur indéterminée ! */
```

Si `p_ok == NULL`, alors on ne cherche pas à l'utiliser !

Pointeurs et notation

Attention aux confusions :

- on déclare un pointeur en utilisant la notation **type** ∗**p**
- on accède à l'adresse correspondante grâce à **p**
- on accède à la valeur correspondance grâce à ∗**p**

Pointeurs : un peu de gymnastique

À votre avis, que fait :

```
1 main()
2 {
3   int i = 3, j = 6;
4   int *p1, *p2;
5   p1 = &i;
6   p2 = &j;
7   *p1 = *p2;
8 }
```

Pointeurs : un peu de gymnastique

À votre avis, que fait :

```
1 main()
2 {
3   int i = 3, j = 6;
4   int *p1, *p2;
5   p1 = &i;
6   p2 = &j;
7   p1 = p2; // c'est ici que ça a changé !
8 }
```

Le passage de paramètres **par valeur**

Exemple de chose à ne pas faire :

```
1 void echange_faux(uint32_t a, uint32_t b)
2 {
3     uint32_t t = a;
4     a = b;
5     b = t;
6 }
```

Le passage de paramètres **par valeur**

Exemple de chose à ne pas faire :

```
1 void echange_faux(uint32_t a, uint32_t b){
2
3         uint32_t t = a; // t reçoit la valeur de la copie de a
4         a = b; // la copie de a reçoit la valeur de la copie de b
5         b = t; // la copie de b reçoit la valeur de t
6
7 }
```

À l'appel de la fonction, de nouveaux espaces mémoire sont alloués pour les paramètres.
On y stocke les valeurs des paramètres mais ce sont des COPIES.

Le passage de paramètres **par valeur**

Exemple de chose à ne pas faire :

```
1 void echange_faux(uint32_t a, uint32_t b){
2
3         uint32_t t = a; // t reçoit la valeur de la copie de a
4         a = b; // la copie de a reçoit la valeur de la copie de b
5         b = t; // la copie de b reçoit la valeur de t
6
7 }
```

À l'appel de la fonction, de nouveaux espaces mémoire sont alloués pour les paramètres.
On y stocke les valeurs des paramètres mais ce sont des COPIES.

Comment s'en assurer (`echange_faux.c`) ?

Le passage de paramètres **par valeur**

Exemple de chose **à faire** :

```
1 void echange(uint32_t *a, uint32_t *b)
2 {
3
4     uint32_t t = *a; // t reçoit la valeur pointée par la copie de a
5     *a = *b; // la valeur pointée par la copie de a reçoit la valeur
          pointée par la copie de b
6     *b = t; // la valeur pointée par la copie de a reçoit t
7
8 }
```

ici, les valeurs passées sont les adresses des paramètres

Utilisation très fréquente des pointeurs

Les chaînes de caractère : `test_chaine.c`

```c
#include <stdio.h>

void main()
{
  char *chaine = "avion";

  printf("adresse de la chaîne : ?? \n", ??);
  printf("adresse du second caractère de la chaîne : ?? \n", ??);
  printf("premier caractère de la chaîne : ?? \n", ??);
}
```

Sources

- Cours de Pablo Rauzy (lien)
- Cours de Jean-Pascal Palus (lien)
- Cours d'Anne Canteaut