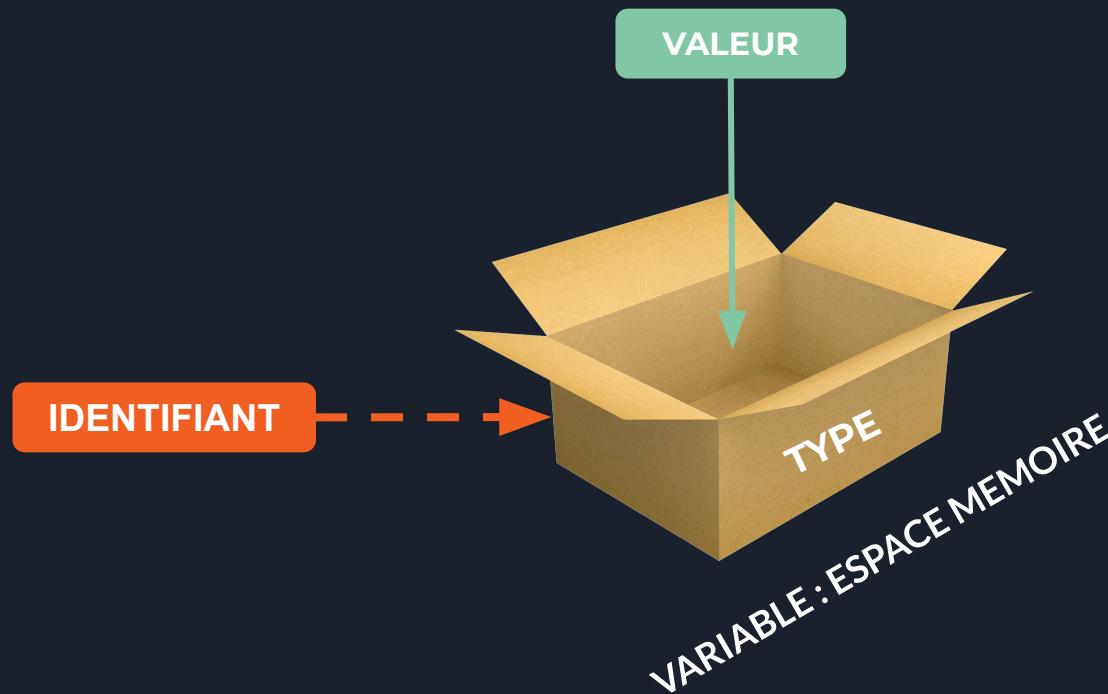


Pointeurs et Références

Mémoire



Mémoire

Tiens **Fonction A**, voici
une boîte à chapeau
nommée **chapi**.

Codeur

Fonction A, peux-tu
mettre ce chapeau dans
la boîte **chapi** ?

Codeur

Fonction B, peux-tu
utiliser le contenu de la
boîte **chapi** ?

Codeur

Oh la belle boîte ! Je
vais la ranger dans notre
entrepôt.

Fonction A

Oh le joli chapeau ! Tout
de suite patron.

Fonction A

WTF ?

Fonction B

Mémoire

Fonction A, où as-tu
rangé la boîte chapi ?

Codeur

Cette boîte est rangée
allée D, rang 42 patron.

Fonction A

Fonction B, peux-tu
récupérer le contenu de
la boîte à chapeau :
allée D, rang 42 ?

Codeur

Ok mec !

Fonction B

Mémoire

Considérons la mémoire comme une bande d'octets.
Un `int` est codé sur **4 octets**.



Occupation mémoire d'une
valeur de type `int`

Adresse

Chaque octet possède une adresse mémoire stockée sur **64 bits** (8 octets).



Adresse

L'**adresse mémoire** de début d'une variable et son **type** permettent de définir l'espace mémoire qu'une variable occupe dans la **RAM**.





Adresse

L'**opérateur de référencement &** devant une variable permet de récupérer son adresse mémoire.

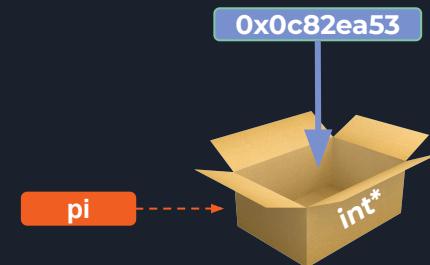
L'adresse mémoire **est une valeur** !

```
int i = 10;  
printf("%p\n", &i); //0C82EA53
```

Pointeur

Le type de valeur **adresse mémoire** est : **[type]***. Une variable stockant une adresse mémoire est appelée **pointeur**.

```
int i = 10;  
int *pi;  
pi = &i;  
printf("%p\n", pi); //0C82EA53
```



Pointeur

On peut travailler sur la variable pointée en utilisant
l'opérateur d'indirection * sur la valeur **adresse mémoire**.

```
int i = 10;  
int *pi;  
pi = &i;  
(*pi) = 5  
printf("%d\n", i); // 5
```

(int*) 0x0c82ea53



Pointeur

Attention à ne pas confondre

i



0x0c82ea53

5

**Identifiant
de variable**

**Variable
(espace mémoire)**

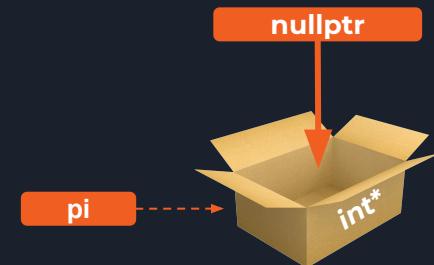
**Adresse mémoire
de variable**

**Valeur de la
variable**

nullptr

Un pointeur dont la valeur est **nulle** ne pointe vers aucune adresse mémoire.

```
int i = 10;  
int *pi = &i;  
pi = nullptr;  
(*pi) // Erreur !
```





Fonctions et pointeurs

Une valeur **adresse mémoire** peut être utilisée en **paramètre de fonction** pour indiquer à une fonction l'emplacement d'une variable de la fonction appelante.



Fonctions et pointeurs

```
int main() {  
    ➔ int a = 10;  
    Increment(&a);  
    return 0;  
}
```

```
void Increment(int* addr) {  
    (*addr)++;  
}
```

a = 10

Fonctions et pointeurs

```
int main() {  
    int a = 10;  
    ➔ Increment(&a);  
    return 0;  ↑  
}  
  
void Increment(int* addr) {  
    (*addr)++;  
}
```

addr = 0x048ef53a (= &a)

a = 10

&a permet de récupérer l'adresse mémoire de la variable. Elle est envoyée en paramètre de la fonction **Increment**.

Fonctions et pointeurs

```
int main() {  
    int a = 10;  
    Increment(&a);  
    return 0;  ↑  
}  
  
void Increment(int* addr) {  
    → (*addr)++;  
}
```

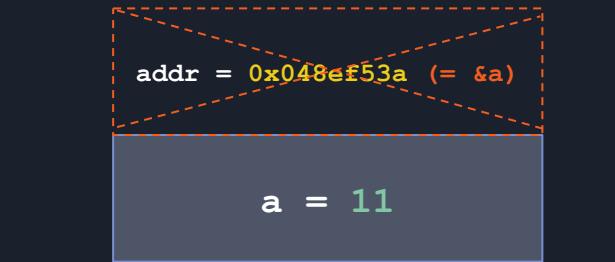
addr = 0x048ef53a (= &a)

a = 11

*addr permet de modifier l'espace mémoire correspondant à l'adresse mémoire addr

Fonctions et pointeurs

```
int main() {  
    int a = 10;  
    Increment(&a);  
    return 0;  
}  
  
void Increment(int* addr) {  
    (*addr)++;  
} ➔ }
```



La variable **a** de **main** a été modifiée par la méthode **Increment**



Pointeurs et tableaux

Un tableau est **un pointeur** vers un espace mémoire contenant ses variables :

```
int tab[3] = {0,1,2}; // tab est un int*
printf("%p\n", tab); //0C82EA53
```

Les crochets [et] servent **d'opérateurs d'indirection ***.

```
tab[0] == (*tab)
tab[1] == (* (tab+1))
```



Référence

Une **référence** ou **alias** est un synonyme **d'identifiant de variable**.

Toute opération appliquée sur la **référence** est appliquée sur la **variable**.



Référence

Une **référence** se déclare avec **[type]&** et est initialisé sur la même ligne d'instruction.

```
int i = 5;  
  
int &j = i;  
  
j = 10;  
  
printf("%d\n", i); // 10
```



Référence

La **référence** et **l'identifiant** ayant servi à l'initialiser identifient la **même variable** en mémoire.

```
int i = 5;
```

```
int &j = i;
```

alors

```
(&i) == (&j)
```



Référence

Contrairement à un pointeur, une **référence** :

- Doit **obligatoirement être initialisé** sur la ligne où elle est déclarée.
- Ne peut **pas être nulle**.
- Ne peut pas être modifié pour pointer sur une autre variable.
- N'a **pas besoin de *** pour référer à la variable.



Référence

En interne, une **référence** est une variable contenant une **adresse mémoire** et fonctionne comme un **pointeur**.

C'est donc une syntaxe **simplifiée** et **sécurisée** du **pointeur**.



Référence

Une **référence** peut être utilisé en **paramètre de fonction** :

```
int main() {  
    int a = 10;  
    Increment(a); // Passage implicite de a en référence  
    return 0;  
}  
  
void Increment(int &b) {  
    b++;  
}
```

Ici, a et b indiquent la même adresse mémoire.



Référence

Pointeurs et **références** permettent donc de copier des adresses mémoires en paramètres de fonction plutôt de copier des valeurs de variables.

Cela apporte trois avantages majeurs aux fonctions :

- Travailler sur les variables de l'appelant.
- Eviter les copies de données lourdes.
- Rendre possible les valeurs de sortie multiples.