H1: De L'utilité Des Pointeurs

Les pointeurs sont plutôt délicats à comprendre, alors nous allons commencer par bien cerner à quoi ils servent. En tous cas, ils sont absolument **indispensables**, et utilisé très régulièrement.

Prenons donc le problème à l'envers et imaginons un problème que nous allons pouvoir résoudre avec nos pointeurs : **nous souhaitons créer une variable qui renvoie deux valeurs.**

Mais impossible ! Une fonction ne peut renvoyer qu'une valeur ; on ne peut pas demander deux `return` à une fonction.

🖍 Un exemple concret

Admettons que l'on souhaite créer une fonction dans laquelle on rentre des minutes, et que celle-ci nous sorte le temps en minutes découpé d'une part en heures et d'autre part en minutes ; si on rentre 90 minutes, on veut qu'elle nous renvoie 1 heure et 30 minutes.

On crée donc notre fonction `decoupe_minutes` :

```
void decoupeMinutes(int heures, int minutes)
{
    heures = minutes / 60;  // 90 / 60 = 1
    minutes = minutes % 60; // 90 % 60 = 30
}
```

Qu'on essaye ensuite d'appeller dans le `main` :

```c
int main(int argc, char *argv[])
{
    int heures = 0, minutes = 90;

    /* On a une variable minutes qui vaut 90.
    Après appel de la fonction, je veux que ma variable
    "heures" vaille 1 et que ma variable "minutes" vaille 30 */

    decoupeMinutes(heures, minutes);

    printf("%d heures et %d minutes", heures, minutes);

    return 0;
}
```

Et le résultat :

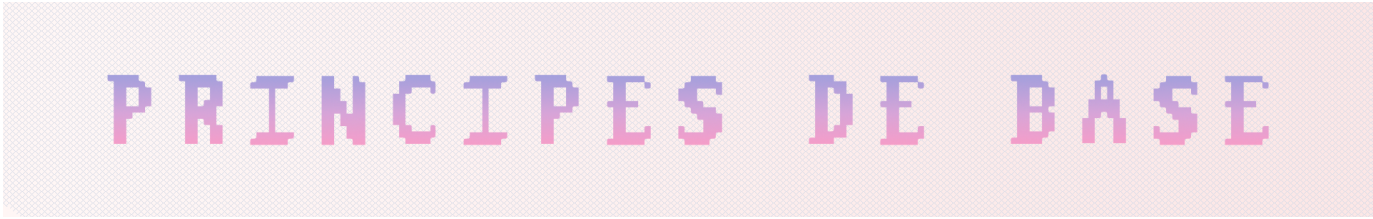
```
0 heures et 90 minutes
```

C'est un échec ! Mais pourquoi ?

📝 Explications

Quand on envoie une variable à une fonction, une copie de la variable est créée : la variable `heure` dans `decoupe_minute` n'est pas la même que dans le `main`. Au sein de la variable, donc, les variables heures et minutes ont les bonnes valeurs, car l'opération écrite a été faite ; cependant, lorsque la fonction se termine, les valeurs des variables sont supprimées ; dans le main, elles valent toujours `0` et `90`.

Sachant qu'on ne peut ni utiliser des variables globales, ni utiliser `static` pour éviter de nous compliquer la vie à la compilation, il nous faut une autre solution : **les pointeurs**.



Pour comprendre comment fonctionnent les pointeurs, on va commencer par reprendre les bases du fonctionnement de la mémoire vive.

La mémoire vive est composé d'un ensemble de cellules. Chacune d'entre elle a son adresse, qui est forcément un nombre (0, 1, 2, 3...). La dernière adresse possible est en général, sur les ordinateurs modernes, un nombre *très* élevé (genre 3 448 765 900 126). Ca fait donc beaucoup de place. Ce nombre d'adresse disponible est défini par la quantité de mémoire dont dispose l'ordinateur.

A chaque adresse, on peut stocker **une** (et on dit bien **une**) valeur, qui est systématiquement un nombre : on rappelle que les ordinateurs fonctionnent en binaire ; ils ne peuvent stocker ni lettres, ni phrases.

Quand on a commencé à vouloir stocké des lettres et des phrases, on a contourné le problème en créant un table de liaison entre chiffres et lettres. Par exemple cette table indique à la machine que "89 représente Y" ; quand la machine rencontre le chiffre "89", elle affiche "Y" à l'écran.

BREF. Revenons à nos pointeurs. 🐑

On se souvient, créer une variable, c'est demander à l'ordinateur de créer une case dans laquelle on va stocker une valeur. Notre système d'exploitation va indiquer au programme quelle cellule de mémoire il peut utiliser, et notre valeur sera donc stockée à une adresse bien précise (et complètement aléatoire), par exemple 1455.

Par exemple, notre variable :

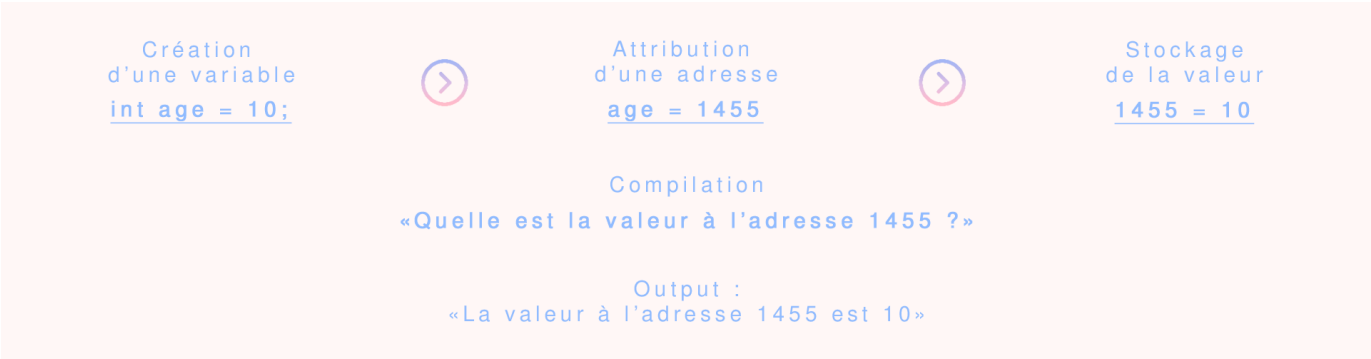
```
int age = 10;
```

A été crée à l'adresse 1455, et sa valeur (10) a été stockée à cette adresse également.

💭 L'affectation de la mémoire est le rôle principal du système d'exploitation. C'est lui qui indique aux programmes où ils peuvent s'installer et la mémoire qu'ils peuvent exploiter.

Lorsqu'on compile, le mot `age` est remplacé par son adresse **1455**, et ce pour chaque occurence du mot. En gros, le compilateur dit à la machine "Dis moi donc ce qui est stocké à l'adresse suivante !" et la machine répond "Oui avec plaisir", et nous sort la valeur de la variable `age` qui était stockée en mémoire.

En bref



💭 On sait afficher la valeur de `age`, suffit la récupérer avec `printf` et `%d, age`. Mais il est possible d'en faire de même pour **l'adresse** de la variable avec `%p, &age`.

On note qu'on ajoute un `&` devant la variable pour faire sortir son adresse.

On a donc accès à deux informations :

- `age`**, soit la valeur de la variable**, qui permet à l'ordinateur de lire la variable en mémoire et de renvoyer **la valeur**,
- `&age`, **soit l'adresse de la variable**, qui demande à l'ordinateur de nous communiquer **l'adresse** de la variable `age`.

UTILISATION

Jusqu'à présent, nous créions des variables pour contenir des valeurs. Maintenant, nous allons créer des variables qui contiennent des adresse : **les pointeurs.**

Pour créer un pointeur, on ajoute simplement `*` devant le nom de notre variable.

Par exemple :

```
int *mon_pointeur = NULL;
```

⚠️ Il faut être vigilant sur deux points :

1. On peut noter `int* mon_pointeur;` pour créer un pointeur, mais on préférera la première syntaxe car elle nous permet d'enchaîner les déclarations : `int *pointeur1, *pointeur2, *pointeur3;`
2. On remarque l'apparition d'un nouvel mot-clé, `NULL`. Il permet, en gros, d'initialiser la valeur par défaut à 0. Il faudra bien le noter **en majuscule**. C'est important d'initialiser notre pointeur en `NULL` car cela permet de nous assurer qu'il ne contient aucune adresse.

Lorsqu'on crée un pointeur, il s'agit donc d'une cellule vide. On va lui attribuer une valeur, et celle-ci sera **l'adresse** d'une autre variable.

🖌️ Exemple

```
int age = 10;
int *pointeurSurAge = &age;
```

📝 Ici, il se passe deux choses :

1. On crée la variable `age`, dont la valeur est 10 ;
2. On crée une variable pointeur de type `int`, dont la valeur est **l'adresse** de la variable `age`.

⚠️ Remarquons qu'il n'existe pas de type de variable `pointeur`. On continue à utiliser les types de variables qu'on connaît déjà, en s'assurant que le pointeur est de même type que la variable créée :

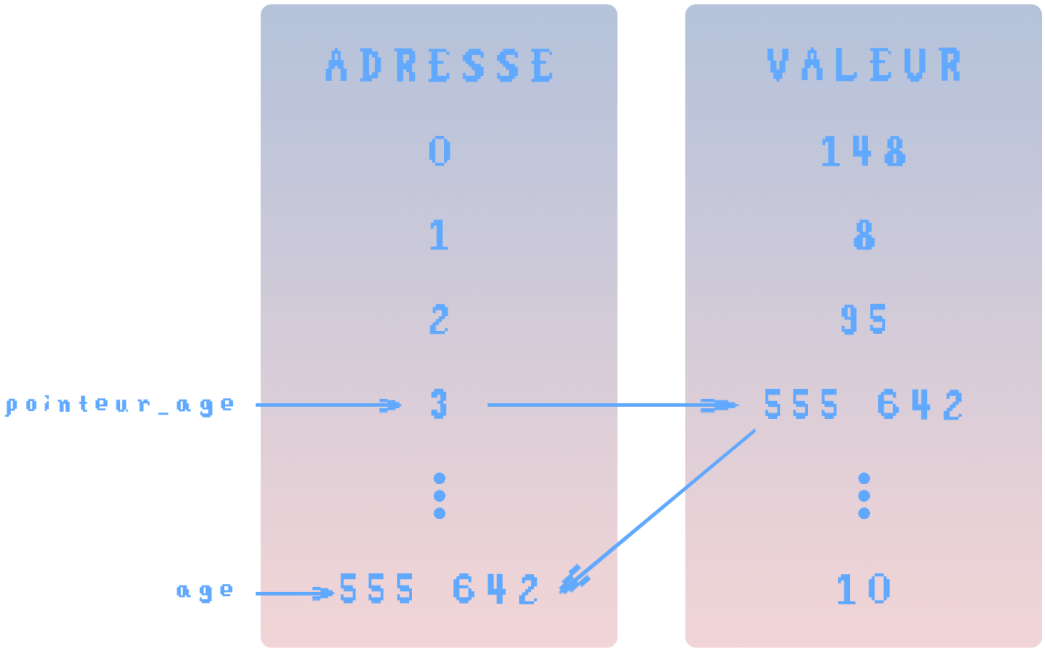
```
int age = 10:
int *pointeur_age = &age;
```

mais en revanche :

```
double hauteur = 9.50;
double *pointeur_hauteur = &hauteur;
```

💡 On dit que qu'un pointeur ***pointe*** sur la variable associée, par exemple, `*pointeur age` pointe sur `age`.

Un petit schéma pour bien que tout soit clair :



- On voit que la varaible `age` est à l'adresse 555 642 et que sa valeur est 10 ;
- Le pointeur `pointeur_age` a été placé à l'adresse 3 (complètement au pif), et sa valeur est 555 642, soit l'adresse de `age`.

reprendre ici