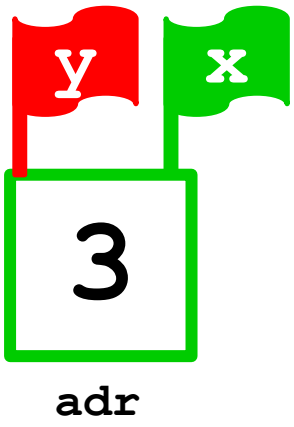
# Déclaration et affectation des références et pointeurs

*par exemple… (toutes opérations sont sur la pile)*

```
double x = 3;
```

## Références

```
double &y = x;
```



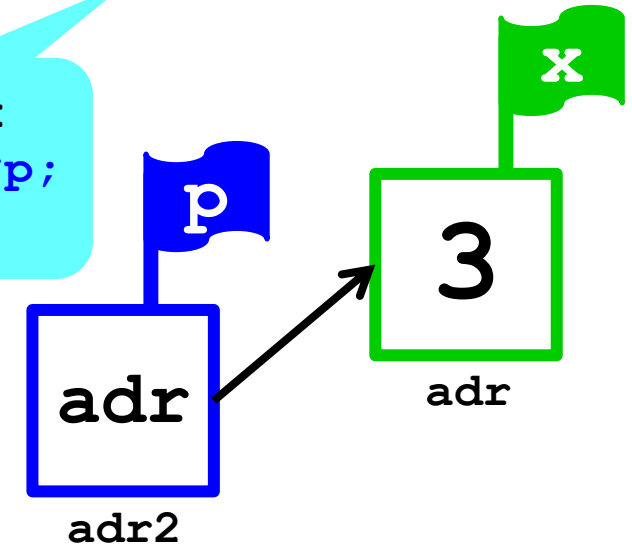

**y** est un alias (synonyme) pour **x**.
En modifiant **y**, on modifiera **x**.
(voir *passage par référence*)

## Pointeurs

```
double *p = &x;
```

également :

```
double *p;
p = &x;
```



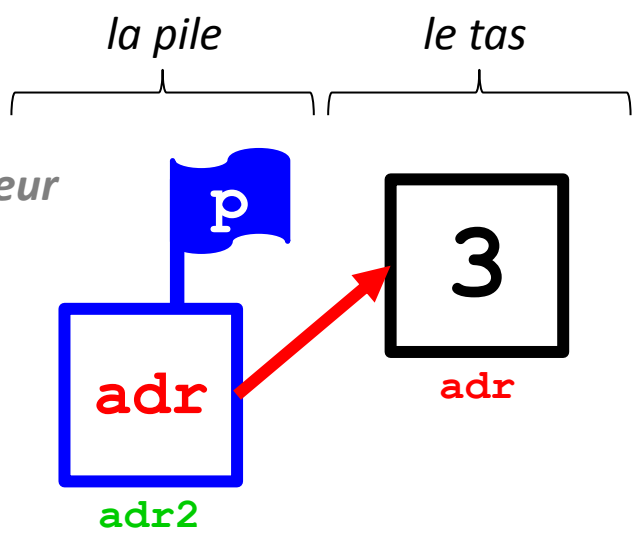

la variable **p** (qui est un pointeur au **double**) contient l'adresse mémoire de **x**

# Faites très attention!


| | l'opérateur de référence **&** | l'opérateur de déréférencement ***** |
|---|---|---|
| **dans une déclaration** *(immédiatement après un type de variable)* | "Je veux déclarer une référence à une variable de ce type" | "Je veut déclarer un pointeur à une variable de ce type" |
| | `double &y` | `double *p` |
| **autrement...** | "Donnez-moi l'adresse mémoire de cette variable" | "Donnez-moi le valeur de la variable qui existe là dans la mémoire" |
| | `&x` | `*p` |
| | `*&x` est le même chose que `x` | `&*p` est le même chose que `p` |

utilisez `p=&x` pour affecter un valeur!

# Introduction à new et delete

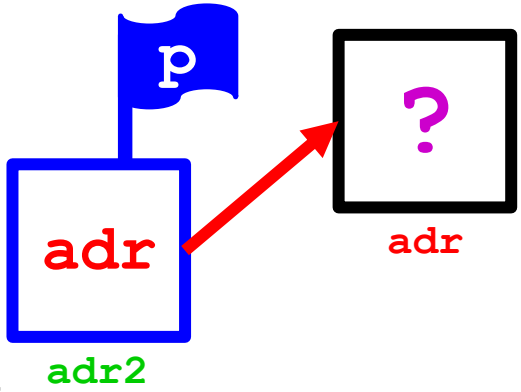



étape 1 : *allouez de la mémoire sur le tas, et affectez y une valeur*

```
double p = new double(3);
//on a créé un double et on y a affecté 3
//il n'a pas de nom (il n'a pas de drapeau!)
//on peut utiliser *p à la place
//(le nom de son pointeur p, avec
// l'opérateur de déréférencement *)
```

étape 2 : *utilisez-la!*

par exemple, **cout << *p** affichera **3** à l'écran,
**cout << p** affichera **adr** à l' écran,
et ***p = 4** affectera **4** au **double** à **adr**




étape 3 : *libérez la mémoire*

```
delete p;
//adr a été libéré
//le programme peut réutiliser cette partie de la mémoire
```

jusqu'ici, la variable était accessible en dehors de toute question de portée, en utilisant ***p**

étape 4 : *réinitialisez le pointeur*

```
p = 0;
//p est toujours déclaré
//mais il ne point plus à adr
//pas de danger d'utiliser *p par erreur
```