Les 3 usages de **static** (1)

1) **static** associé à une **variable locale**

- Portée limitée au bloc de sa déclation
- Durée de vie permanente
- N'est initialisée qu'une seule fois
- si la valeur initiale n'est pas précisée,
  - -> est initialisé avec des 0
  - -> ou par le constructeur par défaut

⇒ Conserve sa valeur d'un appel au suivant

⇒ Privilégier cet usage de **static** par rapport aux deux autres car ***c'est le plus local***

Ex2: si on veut l'exploiter pour la variable mémorisant l'état courant de la lecture d'un fichier (cf série précédente niveau0), il faut prévoir un mécanisme de ré-initialisation.

```
...
unsigned int nb_appel()
{
    static unsigned count(0);
    return ++count;
}

int main()
{
    cout << nb_appel() << endl;
    cout << nb_appel() << endl;
    cout << nb_appel() << endl;
    …
```

Affiche :

```
1
2
3
```

Les 3 usages de `static` (2)

2) **static** associé à une **variable globale** rend cette variable ***confidentielle au module***

- Portée limitée au module = *espace de noms non-nommé*
- Durée de vie permanente
- Accessible par toutes les fonctions du module
- n'est initialisée qu'une seule fois
- si la valeur initiale n'est pas précisée,
  -> est initialisé avec des 0
  -> ou par le constructeur par défaut

- Ne JAMAIS mettre dans l'interface d'un module
- Utile pour des constantes locales au module
- Sinon, ***à limiter au strict nécessaire***
  OK pour petit module mono-classe
  Ex : ensemble des instances d'une classe,
  reste caché dans l'implémentation ->

myclass.cc

```
...
#include "myclass.h"

static vector<MyClass> tab;// vide

// externalisation de la définition
// des méthodes de la classe MyClass

MyClass::Myclass() ...
{...}

...
```

Aussi utilisable pour une fonction

Les 3 usages de `static` (3)

2) **static** associé à un **attribut** partage la valeur de cet attribut avec toutes les instances

- Portée limitée à la classe
- Durée de vie permanente ; pas besoin d'instance
- Accessible par toutes les méthodes de la classe
- DOIT être initialisé en dehors de la classe
- si la valeur initiale n'est pas précisée,
  -> est initialisé avec des 0
  -> ou par le constructeur par défaut

- Est visible dans l'interface d'un module
- NE PAS RENDRE **`public`**
- Utile pour des paramètres communs
- Sinon, à limiter au strict nécessaire

Aussi OK pour gérer l'ensemble
des instances d'une classe si
le but de cette classe est de gérer
un seul ensemble d'instances.

Aussi utilisable pour une méthode

myclass.h

```
...
class MyClass {
private:
    static vector<MyClass> tab;
...
```

myclass.cc

```
...
#include "myclass.h"

vector<MyClass> MyClass::tab;// vide

// externalisation de la définition
// des méthodes de la classe MyClass

MyClass::Myclass() ...
{...}
...
```