

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

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

- Portée limitée au bloc de sa déclaration
- 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() ...
{ ... }

...
```

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.

myclass.h

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

```

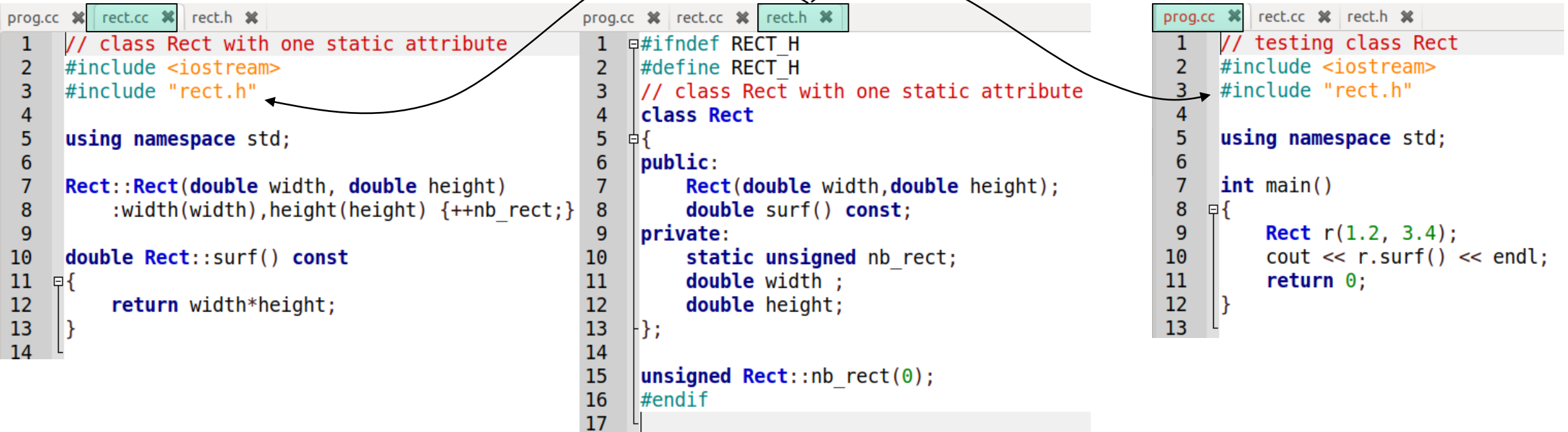
myclass.cc

```
...  
#include "myclass.h"  
  
vector<MyClass> MyClass::tab; // vide  
  
// externalisation de la définition  
// des méthodes de la classe MyClass  
  
MyClass::Myclass() ...  
{...}  
...  

```

Quizz1

Soit le module **rect** qui est testé dans la fonction main du fichier **prog.cc** :



```
1 // class Rect with one static attribute
2 #include <iostream>
3 #include "rect.h"
4
5 using namespace std;
6
7 Rect::Rect(double width, double height)
8     :width(width),height(height) {++nb_rect;}
9
10 double Rect::surf() const
11 {
12     return width*height;
13 }
14
```

```
1 #ifndef RECT_H
2 #define RECT_H
3 // class Rect with one static attribute
4 class Rect
5 {
6 public:
7     Rect(double width,double height);
8     double surf() const;
9 private:
10    static unsigned nb_rect;
11    double width ;
12    double height;
13 };
14
15 unsigned Rect::nb_rect(0);
16 #endif
17
```

```
1 // testing class Rect
2 #include <iostream>
3 #include "rect.h"
4
5 using namespace std;
6
7 int main()
8 {
9     Rect r(1.2, 3.4);
10    cout << r.surf() << endl;
11    return 0;
12 }
13
```

Donner la réponse correcte si on lance : `g++ -std=c++11 prog.cc rect.cc -o prog`

- A. Erreur de **compilation** parce que l'attribut static est défini deux fois
- B. Erreur de **l'édition de liens** parce que l'attribut static est défini deux fois
- C. L'attribut static est défini deux fois ; cependant un exécutable est produit et fonctionne bien
- D. L'attribut static est défini une seule fois ; l'exécutable est produit et fonctionne bien