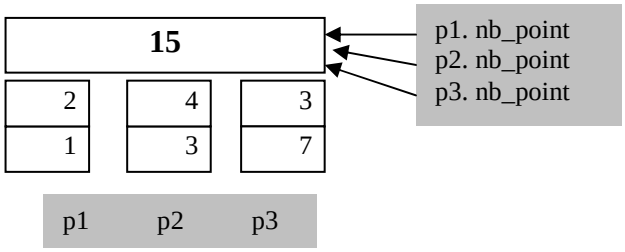
Rappel sur le qualificatif « *static* »

Le qualificatif « **static** » permet de créer des données membres globales qui n'existeront qu'en un seul exemplaire indépendamment du nombre d'objets créés (même valeur pour tous les objets de la classe).
Une données membres "**static**" existe même si aucun objet de la classe n'a été créé.
L'accès à cette variable peut se faire en utilisant l'opérateur de résolution de portée. Elle doit être initialisée en dehors de tout corps de fonction.

Représentation graphique :




```
class point
{
public :
        int static nb_point;
        float cx,cy;

        voir( ) ;
};
point::nb_point = 15;

main ( )
        {
        point p1 ( 1 , 2 );
        point p2 ( 3 , 4 );
        point p3 ( 7 , 3 );
        }
```

Appliqué à une variable.

Ex :la classe point.

Fichier déclaration:

```
static  int     nb_point_crees ;      //variable globale commune à tous les objets qui seront déclarés par la suite.
static  const  int  taille;           //idem pour cette constante, elle aura la même valeur pour tous les objets.
```

Fichier définition

```
int point:: nb_point_crees = 0;       //initialisation des données membres statiques (globales)
int const  point::taille = 12;

main()
      //correct, le compilateur sait de quelle variable il s'agit
      //malgré l'absence de déclaration d'objets de types « point ».
      cout << point : : nb_point_crees;

      //incorrect, le compilateur ne reconnaît pas la variable
      // cout << nb_point_crees;
```

Appliqué à une fonction.

Permet d'exécuter des méthodes d'une classe lorsqu' aucun objet n'existe.
Permet de créer des objets.

Fichier déclaration :

```
static  int      visu_nb_point_crees ( );//déclaration de la fonction « static »
```

Fichier définition :

```
//corps de la fonction NB :opérateur de résolution de portée
int point:: visu_nb_point_crees ( )
{
```

```
  cout <<"\n-------------appel de visu_nb_point_crees:" << nb_point_crees ;
}
main()
        {
point p4,p5;
      ….
point:: visu_nb_point_crees ( );          //correct : utilisation avec l'ORP même si
aucun objet n'existe
p4. visu_nb_point_crees ( );              //correct
p5. visu_nb_point_crees ( );
//correct. Les 3 appels donneront le même résultat car la variable nb_point_crees
//a la même valeur pour tous les objets.
```

Le qualificatif « const ».

Appliqué à une variable.

Permet de déclarer une constante.

```
const int i = 4 ;
i = 7 ;                    //rejet du compilateur
```

Appliqué à une fonction.

Permet de définir si une fonction peut s'appliquer à un objet « constant » ou non.

Ex :

```
class vecteur
{
        public :
        int x; int y;
        int   angle( );              //ne peut s'appliquer aux objets constants
        int   module( ) const;       //peut s'appliquer aux objets constants ou non constants
```

Si on écrit un programme tel que :

```
vecteur v1 ;
const vecteur v2 ;

v1.angle() ;              //correct
v2.module ();             //correct
v2.angle() ;              //incorrect car v2 est constant et « angle » ne peut s'appliquer à une constante
v1.module() ;             //correct
```

Le qualificatif « inline ».

Lorsqu'une déclaration de fonction est précédée du mot « inline », le compilateur devra insérer le code correspondant à chaque appel de cette fonction. Cela entraine une perte de place mémoire et un gain de temps d'exécution du programme.
Ce procédé ne se conçoit que si le corps de la fonction occupe peu de place en mémoire.

Ex :

```
inline  int   fct1() ;
main()
{
      cout << …….
//le code objet de la  fonction fct1() sera implanté à chaque appel
//(2 fois) alors que le code de la fonction fct2() ne sera implanté
//qu'une fois. l'accès à fct2() se fera par le mécanisme
// d'appel de sous programme.
      Fct1() ;
      Fct2() ;
      Fct1() ;
      Fct2() ;          …}
```