# Accès Nommage Visibilité

## La syntaxe de l'encapsulation.

Pour assurer l'encapsulation nous voulons que le langage fasse respecter des règles d'accès.

Le respect de ces règles d'accès simplifie énormément le travail de correction d'erreur / correction de code commun à tout les développement informatique.

# Encapsulation des Données

Une bonne stratégie de codage consiste à déclarer que
les fonctions d'accès aux données sont publiques,
et que l'implantation des structures de données est cachée.

Exemple

```
class Pile {
  public : // l'interface fonctionnelle publique
    Pile(int n = 10);
    void push(string x);
    string top();
    string pop();
  private : // les données privées
    int sp;
    string * p;
};
```

# Rappels

**Portée des identificateurs en C**

Les identificateurs se partagent en plusieurs espaces de noms
indépendants déterminés par les objets qu'ils représentent :

* variables, fonctions, noms de types, constantes d'énumération
* labels (pour les branchements)
* labels de structures, unions, et énumérations
* membres de chaque structure ou union individuellement

Dans une unité (resp. bloc), un identificateur est utilisable depuis
sa déclaration jusqu'à la fin du fichier (resp. bloc),
sauf dans les blocs (resp. sous-blocs)
où il est redéclaré dans le même espace de noms

la définition d'un identificateur masque celle du même
identificateur du même espace de noms d'un sur-bloc ou de l'unité

une occurrence d'un identificateur fait référence à sa
déclaration dans le plus petit bloc (ou l'unité) qui les contient

# Classes d'allocation

**Classes d'allocation**

Zone mémoire utilisateur :
zone texte : (code du programme)
zone données : (variables globales)
pile : (données temporaires)
tas : (mémoire dynamique)

Classes d'allocation des variables
**statique** : variables implémentées en zone données
**automatique** : variables implémentées dans la pile
**dynamique** : variables implémentées dans le tas, allouées à la demande

# Les trois catégories d'accès

Les droits d'accès sont accordés aux fonctions membres, ou aux fonctions globales.
L'unité de protection est la classe: tous les objets d'une classe bénéficient de la même protection.
Les attributs de protection suivants sont être utilisés pour les membres d'une classe:

* un membre `public` est accessible à tout fonction;
* un membre `private` n'est accessible qu'aux fonctions membre **de la classe** ou aux fonctions amies;
* un membre `protected` n'est accessible qu'aux fonctions membre de la classe ou des classes dérivées ou aux fonctions amies.

Par défaut, les membres d'une classe sont privés, les membres d'une struct sont tous public.

```
class X {
    int priv; //prive par defaut
  protected:
    int prot;
  public:
    int publ;
    void m();
}
```

La fonction membre `X::m()` a un accès non restreint:

```
void X::m() {
  priv = 1; // ok
  prot = 1; // ok
  publ = 1; // ok
}
```

Une fonction globale n'accède qu'aux membres publics:

```
printf("%d", x.priv); // erreur
printf("%d", x.prot); // erreur
printf("%d", x.publ); // ok
```

Même comportement pour une fonction membre d'une classe qui n'est **ni dérivée** ni amie.

# Fonctions amies

L'amitié permet d'accéder aux parties privées.

Une "déclaration d'amitié" incorpore, dans une classe X, les fonctions autorisées à accéder aux membres privés. Une fonction amie a le même statut qu'une fonction membre de X.

Fonction globale

```
class X { ...
  friend f(X);
}
```

Ici, f est autorisée à accéder aux membres privés de la classe X.

Fonction membre de Y

```
class Y { ...
  void f(X& x);
}
class X { ...
  friend Y::f(X& x);
}
void Y::f(X& x) {
  x.priv = 1; // ok
  x.prot = 1; // ok
  x.publ = 1; // ok
}
```

Une classe entière est une classe amie par

```
class X { ...
  friend class Y;
}
```

# Accès et héritage

Une classe dérivée est elle-même déclarée avec un mode de dérivation private (défaut), protected ou public, par

```
class Yu : public X {};
class Yo : protected X {};
class Yi : private X {};
```

La protection des membres de X se transforme pour les classes dérivées des classes dérivées.

L'accès aux membres de X, par une méthode de la classe dérivée, est la restriction **la plus forte** :

* Une donnée `private` de X est **toujours** inaccessible aux classes dérivées.
* Une donnée `protected` de X a la protection `protected` dans Yu et Yo, et `private` dans Yi.
* Une donnée `public` de X a la protection `protected` dans Yu et `private` dans Yi.

Attention : l'accessibilité concerne uniquement les membres de l'objet lui-même (this), et non pas des paramètres:
* une donnée protégée est accessible en tant que membre de l'objet;
* une donnée d'un objet de la classe de base n'est pas accessible à une fonction membre de la classe dérivée.

```
void Yu::h(X& x) {
  x.publ = 1;
  this -> prot = 2;  //ok
  x.prot = 2; // erreur: membre x.prot inaccessible
}
```