# Bibliothèques et Namespaces

# Bibliothèque : Définition

Une **bibliothèque** est un **code source tiers** ajouté à un projet.

Elle définit un **ensemble de fonctions, de types et de variables** qui vient enrichir le code dans lequel elle est ajoutée.

# Bibliothèque : Définition

Une **bibliothèque** est composée de


**Fichiers sources**

.h .hpp

.c .cpp

**Fichiers compilés**

.lib .dll

.a .so

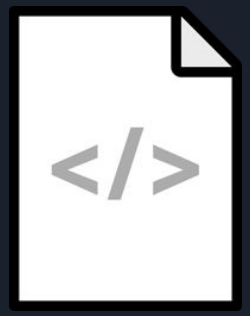
# Bibliothèque : Types

Il existe trois types de bibliothèques :

- Les bibliothèques **non-compilées**
- Les bibliothèques **compilées statiques**
- Les bibliothèques **compilées dynamiques**

# Bibliothèque non-compilée

**Une bibliothèque non-compilée** est composée de





Les sources sont intégrés dans le binaire généré à la compilation.

# Bibliothèque : Fichiers sources

Les **fichiers sources** (.c, .cpp, ...) contiennent la **définition** des fonctions, des types et des variables de la bibliothèque.

Les fonctions, types et variables destinés à être exposés au **code extérieur** sont déclarés dans les **headers de la bibliothèque**.

# Bibliothèque : Header

Un **header de bibliothèque** déclare des fonctions, variables et types exposés au code exterieur.

L'ensemble de ces déclarations est appelé **API** (interface de programmation applicative).

**Inclure un header de bibliothèque** dans notre code permet à ce dernier de **connaître et d'utiliser l'API** de la bibliothèque.

# Bibliothèque : Header

L'inclusion de header se fait en utilisant `#include`. En fonction de l'emplacement de la bibliothèque

```
#include "someLibrary.h"
```

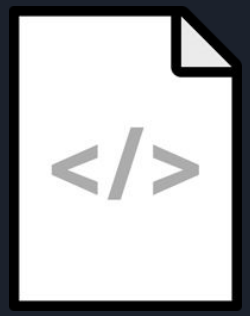
Cherche le header dans le projet puis hors du projet

```
#include <someLibrary>
```

Cherche le header hors du projet

# Bibliothèque statique

**Une bibliothèque compilée statique est composée de**




**Headers**

.h .hpp

**Fichiers compilés statiques**

.lib .a .so

Ils sont inclus dans le binaire à la compilation.

# Bibliothèque : Fichier statique

**Les fichiers compilés statiques** font le **lien** entre la partie non-compilée et la partie compilée.

Dans le cas d'une bibliothèque statique, les fichiers statiques sont les **fichiers sources compilés**.
Ils contiennent la définition des fonctions, variables et types.

# Bibliothèque dynamique

**Une bibliothèque compilée dynamique est composée de**

**Headers**

.h .hpp

**Fichiers compilés statiques**

.lib .a .so

**Fichiers compilés dynamiques**

.dll .dylib

# Bibliothèque : Fichier dynamique

Les **fichiers compilés dynamiques** contiennent la **définition** des fonctions, variables et types.

Ils ne sont **pas inclus** dans le fichier binaire généré à la compilation (**fichiers séparés**).

# Bibliothèque : Fichier dynamique

**Les fichiers dynamiques** doivent être placés dans un répertoire **accessible au binaire généré** pour que le programme puisse fonctionner.

Du fait qu'ils soient externes, **les fichiers dynamiques** peuvent être mis à jour **séparément** du programme qui les utilisent.

# Edition de lien

De manière à permettre au compilateur d'éditer des liens entre fichiers sources et fichier binaires, il est nécessaire d'indiquer **les fichiers statiques** utilisés dans la **configuration du projet**.

# Namespace

Les namespaces permettent d'éviter les conflits entre identifiants.

```
namespace myns {
  int val = 500;
}

namespace myns2 {
  int val = 10; // Deux "val" dans deux namespace différents
}
```

# Namespace

Pour accéder aux variables d'un namespace, on utilise `::`

```
int main() {
  myns::val = 8;
  myns2::val = 12; // Les val viennent de deux
namespace différents.
}
```

# Namespace

`using namespace` permet d'éviter de répéter les noms de namespaces.

```
using namespace myns;

int main() {
  val = 8; // myns::val
}
```

# Namespace

Il est possible d'utiliser `using namespace` tant qu'il n'y a pas d'ambiguïté sur les identifiants.

```
using namespace myns;

int main() {
  val = 8; // myns::val
  myns2::val = 8; // Pas d'ambiguïté
  int val = 0; // Ambiguïté => Erreur !
}
```