

COURS DE C++

6 - Déclarations et Définitions. Code modulaire.

SITUATION ACTUELLE

- Pour le moment on a écrit des programmes consistant en un unique fichier source C++
- Pour pouvoir compiler notre programme le compilateur se contente de lire et d'interpréter ce fichier et d'en extraire un exécutable.
- Nous avons toujours fait en sorte que les fonctions soient définis avant leur utilisation

DÉCLARATIONS DE FONCTIONS

- Le code suivant ne compile pas :

```
void g() { f(); }
void f() {}
```
- On résout cela en intervertissant les définitions
- Des fois cela n'est pas possible, notamment quand f et g ne sont pas définies dans le même fichier
- On utilise alors une déclaration de la fonction. Cela revient à dire au compilateur : cette fonction existe, voila comment l'appeler. C'est une promesse de définition.

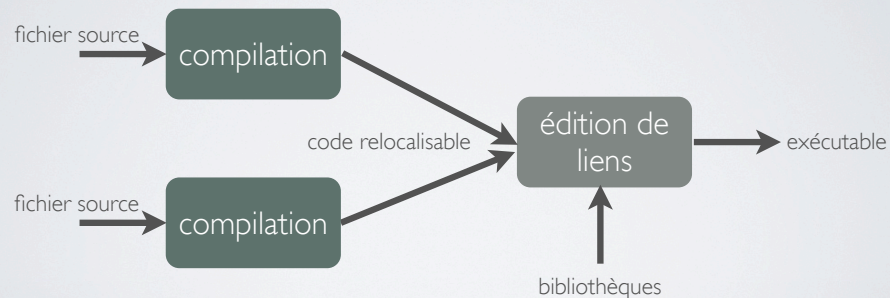
SYNTAXE DE LA DÉCLARATION

- Si on a une fonction de la forme `retT f(arg1T v1, ..., argnT vn) {...}`
- Sa déclaration est `retT f(arg1T, ..., argnT);`
- Pour des raisons de lisibilité et de documentation on peut donner des noms de variables, mais ceux-ci sont ensuite ignorés
- Le code précédent modifié ainsi compile :

```
void f();
void g() { f(); }
void f() {}
```
-

COMPILATION MULTI-FICHIERS

- La compilation s'effectue en fait en deux temps :



En fait la situation est un peu plus complexe (DLLs)

FICHIERS EN-TÊTES

- Pour pouvoir appeler des fonctions définies dans d'autres fichiers il faut avoir accès à leur déclarations
- Recopier dans tous les fichiers les déclarations des fonctions que l'on utilise serait trop long

A la place on met les définitions dans des fichiers en-tête (extension .h) et on inclue ses fichiers : `#include "toto.h"`

CLASH DE NOMS

- Exemple de situations :

vous avez programmé une fonction `priceCall` dont vous vous servez dans vos prototypes

- un client vous demande d'intégrer votre prototype dans son application
- ce même client utilise une fonction du même nom dans son application
- problème : les deux fonctions n'ont rien à voir (pas le même nombre d'arguments, pas la même sémantique, ...)
- vous ne pouvez pas abandonner une des deux fonctions (votre prototype utilise beaucoup la votre et l'application de votre client repose sur l'autre)

ESPACES DE NOMS

- Le C++ vous permet de définir des espaces de noms

Ainsi votre client pourra avoir un espace de nom `sg` et vous un espace de nom `ih`

Les fonctions `priceCall` sont prefixées par leur espaces de noms `sg::priceCall` et `ih::priceCall`

Pour définir des fonctions dans des espaces de noms on encadre les définitions de `namespace ih { ... }`

Pour utiliser la fonction `f` dans l'espace de nom `n` on appelle `n::f`

Pour se placer implicitement dans un espace de noms `n` on utilise `using namespace n;`

La bibliothèque standard du C++ utilise l'espace de noms `std`

UTILISATION DANS ECLIPSE

- Voir Démo.