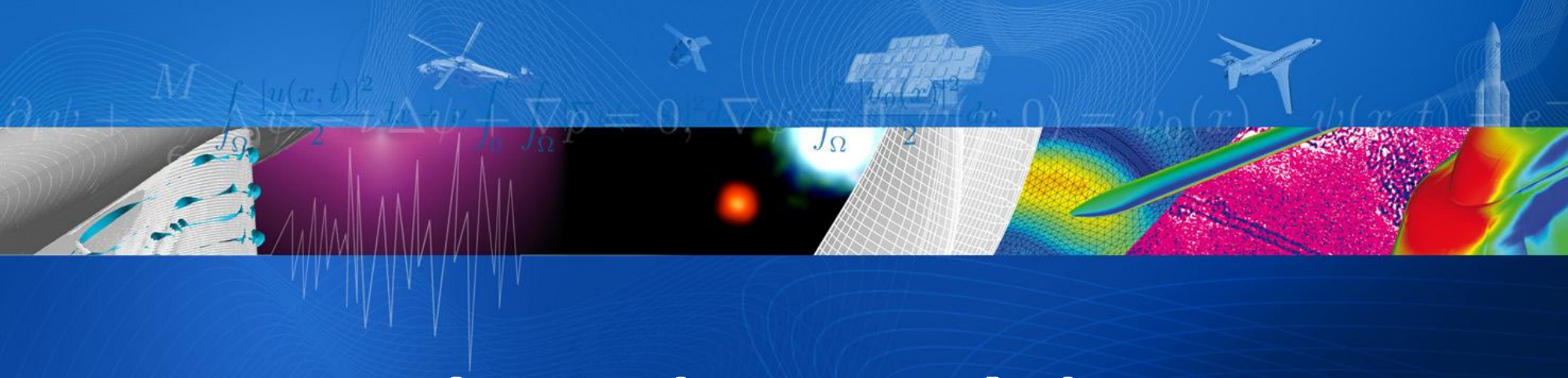


Introduction à la programmation C++

Fichiers séparés / opérateurs

Nicolas Audebert

Mercredi 12 octobre 2016

ONERA
THE FRENCH AEROSPACE LAB

retour sur innovation

Plan de la séance

Jusqu'à maintenant ...

- un `main` (= point d'entrée du programme),
- des fonctions,
- des structures,

... mais tout dans le même fichier.

Plusieurs fichiers pour ...

1. mieux organiser son code (plus lisible, regrouper par modules),
2. partager ses fonctions entre plusieurs projets,
3. accélérer la compilation (notamment pour les gros projets)

Plusieurs fichiers sources

Fonctions dans deux fichiers :

C++

```
// fichier1.cpp


A f1(B b){
    ...
    g2(var1); // Erreur
    ...
}
```

C++

```
// fichier2.cpp


void g2(C c){
    ...
}
```

Attention

Pour utiliser une fonction, il faut qu'elle soit connue dans le fichier où on l'utilise.

Plusieurs fichiers sources 2

Fonctions dans deux fichiers :

C++

```
// fichier1.cpp

void g2(C c);

A f1(B b){
    ...
    g2(var1); // OK
    ...
}
```

C++

```
// fichier2.cpp


void g2(C c){

    ...

}
```

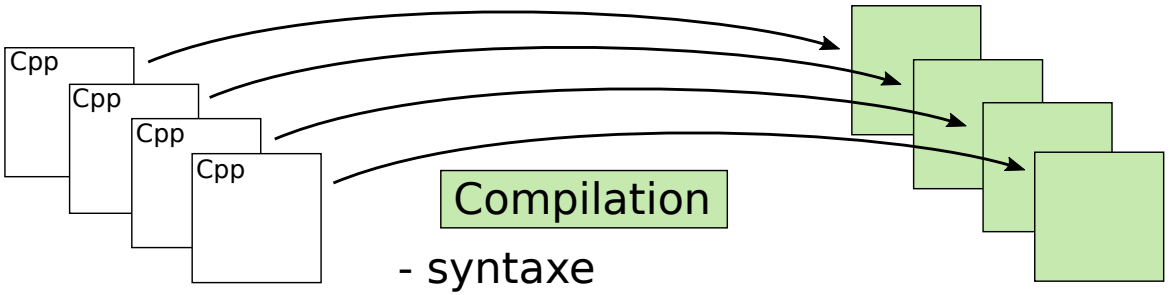
Solution

On déclare la fonction dans le fichier qui utilise la fonction pour dire au compilateur que la fonction existe.

Pourquoi ?

Il y a deux étapes dans pour la production de l'executable.

- La compilation (fichiers objets)



Compilation

- syntaxe
- compatibilité des types
- déclaration des fonctions

...

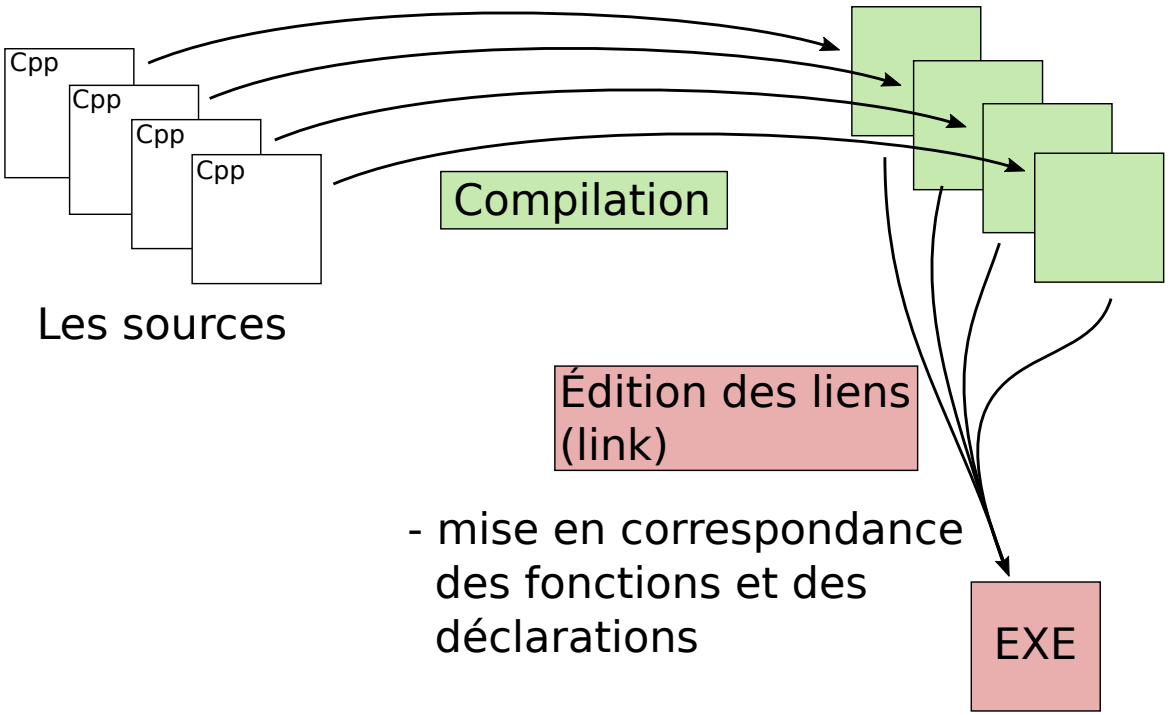
Un produit par .cpp

Les sources

Pourquoi ?

Il y a deux étapes dans pour la production de l'executable.

- La compilation (fichiers objets)
- L'édition des liens (executable)



Pourquoi ? 2

Un fonctionnement en deux étapes pour :

- compiler plus rapidement : on ne recompile que les fichiers modifiés
- possibilité de faire des librairies : fichiers précompilés et appelé dans le code (exemple Imagine++)

CMake

CMakeList.txt

```
CMAKE_MINIMUM_REQUIRED(VERSION 2.6)
FILE(TO_CMAKE_PATH "$ENV{IMAGINEPP_ROOT}" d)
IF(NOT EXISTS "${d}")
MESSAGE(FATAL_ERROR "Error: environment variable
      IMAGINEPP_ROOT=" "${d}")
ENDIF(NOT EXISTS "${d}")
SET(CMAKE_MODULE_PATH ${CMAKE_MODULE_PATH} "${d}/CMake")
FIND_PACKAGE(Imagine)

PROJECT(Exemple_Cours)

add_executable(Exemple_exec
 fichier1.cpp fichier2.cpp fichier3.cpp ...
 )
ImagineUseModules(Exemple_exec Graphics)
```

Ajouter un fichier dans un projet existant

Solution pour tous les IDEs

1. créer le fichier dans le même dossier que les autres,
2. modifier le CMakeLists.txt avec un éditeur de texte : ajouter le nom du fichier
3. recompiler le programme dans l'IDE.

Ajouter un fichier dans un projet existant

Solution pour QtCreator

Dans QtCreator :

1. ouvrir le menu `File/New File or Project` ou faire `Ctrl+N`, choisir C++ Source File. **Attention : mettre le fichier dans le dossier des sources**,
2. rajouter ce fichier dans le CMakeLists.txt,
3. recompiler le programme dans QtCreator.

Les fichiers d'entête

Constat

- Lourd de recopier toutes les déclarations
- Pas de partage des structures

Solution

Mettre toutes les déclarations et les structures dans des fichiers d'entête (header) repérés par l'extension **.h**

Implémentation

C++

```
// fichier1.cpp

#include "fichier2.h"

A f1(B b){
    ...
    g2(var1);
    ...
}
```

C++

```
// fichier2.h
void g2(C c);

struct Vect{
    ...
};
```

C++

```
// fichier2.cpp

//habitude a prendre
#include "fichier2.h"

void g2(C c){...}
```

Note

Jusqu'à présent, les fonctions externes au projet étaient incluses grâce à la directive `#include<...>`. Il s'agit en fait de headers externes pour différents librairies (librairie standard `std`, librairie Imagine++, ...).

En pratique

Méthode générique

Exactement comme pour les `.cpp`.

QtCreator

Idem, mais il faut créer un C++ Header File.

Inclusions mutuelles

Faire un `include` fait un copier/coller du header dans le fichier source.
Rien n'interdit les inclusion mutuelles :

C++

```
// fichier1.h
#include "fichier2.h"
A f1(B b);
```

C++

```
// fichier2.h
#include "fichier1.h"
void g2(C c);
```

Attention
Boucle dans les inclusions $\rightarrow$ compilation impossible et plantage.

Inclusions mutuelles 2

Pour tous les OS :

C++

```
// fichier1.h

#ifndef NOM_UNIQUE
#define NOM_UNIQUE

#include "fichier2.h"

A f1(B b);

#endif
```

Plus simple en utilisant la directive `pragma` (plus récente) :

C++

```
// fichier1.h

#pragma once

#include "fichier2.h"

A f1(B b);
```

Plan de la séance

Opérateurs

Les opérateurs définissent le comportement de certains signes de ponctuation ou mathématiques :

- +,-,/,*,= ...

Il est possible de redéfinir ces opérateurs pour les utiliser avec les structures que l'on a créées.

Exemple

C++

```
struct Vect{
   double x,y;
};
```

C++

```
Vect v1, v2;
...
// additionner deux vecteurs
Vect v3 = {v1.x+v2.x, v1.y+v2.y};
Vect v4;
v4.x = v1.x+v2.x;
v4.y = v1.y+v2.y;
// produit scalaire
double s = v1.x*v2.x + v1.y*v2.y;
```

Ce qu'on voudrait :

C++

```
Vect v1, v2;
...
// additionner deux vecteurs
Vect v3 = v1+v2;
// produit scalaire
double s = v1*v2;
```

Implémentation

C++

```
struct Vect{
   double x,y;
};

// operateur + sur des vecteurs
Vect operator+(Vect vA, Vect vB){
    Vect v = {vA.x+vB.x, vA.y+vB.y};
    return v;
}

// operateur * sur des vecteurs
double operator*(Vect vA, Vect vB){
    return vA.x*vB.x + vA.y*vB.y;
}
```

C++

```
Vect v1, v2;
// On peut utiliser les nouveaux
// operateurs normalement

// additionner deux vecteurs
Vect v3 = v1+v2;

// produit scalaire
double s = v1*v2;
```

Surcharge des opérateurs

C++

```
// operateur * pour deux vecteurs
double operator*(Vect vA, Vect vB){
    return vA.x*vB.x + vA.y*vB.y;
}

// operateur * vecteur et reel
Vect operator*(Vect vA, double alpha){
    Vect v = {alpha*v.x, alpha*v.y};
    return v;
}
```

C++

```
Vect v1, v2;
...

// produit scalaire
double s = v1*v2;

// multiplication par un reel
double m = 5.5;
Vect v3 = v1*m;
```

Surcharge des opérateurs 2

Attention

L'ordre des arguments et important :

`v1 * m` est différent de `m * v1`

C++

```
// operateur * vecteur et reel
Vect operator*(Vect vA, double alpha){
    Vect v = {alpha*v.x, alpha*v.y};
    return v;
}

// operateur * vecteur et reel
Vect operator*(double alpha, Vect vA){
    return v*alpha;
}
```

C++

```
Vect v1, v2;
...

// multiplication par un reel
double m = 5.5;

Vect v3 = v1*m;

Vect v4 = m*v2;
```

Plan de la séance

Le TP du jour

On prend le même et on continue :

1. Finir le TP de la séance précédente
2. Incrémenter en utilisant plusieurs fichiers
3. Utiliser des opérateurs personnalisés pour les calculs