

# PRALG: rappels de C++

Pascal Monasse  
pascal.monasse@enpc.fr

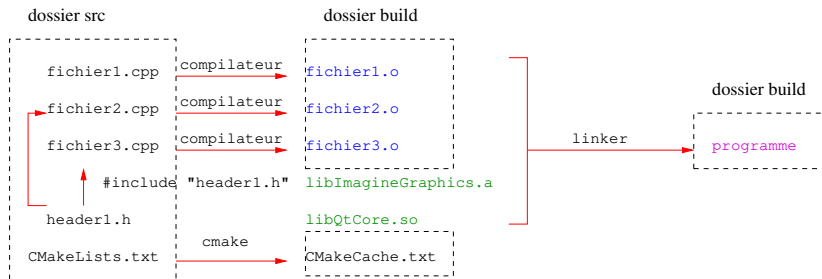


ÉCOLE NATIONALE DES  
**PONTS**  
ET **CHAUSSEES**



**IP PARIS**

# Build



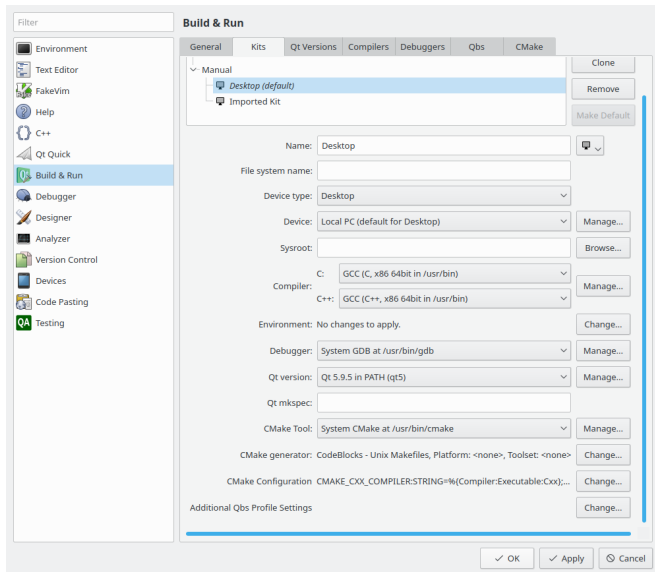
## CMakeLists.txt

```
cmake_minimum_required(VERSION 3.10)
project(programme)
find_package(Imagine REQUIRED COMPONENTS
Graphics)
add_executable(programme fichier1.cpp fichier2.cpp
fichier3.cpp header1.h)
target_link_libraries(programme Imagine::Graphics)
```

## Compilateurs :

- ▶  gcc (GNU compiler collection)
- ▶  mingw32 (Minimal GNU for Windows)
- ▶  clang

# Menu “Tools/Options” de QtCreator



Les outils  
(cmake,  
compilateur,  
version de  
Qt) sont  
rassemblés  
dans un kit  
de  
QtCreator

# Problèmes divers

The screenshot displays a CMake IDE interface. The 'Projects' sidebar on the left shows a project named 'Project' with a 'CMakeLists.txt' file and a 'programme' directory containing 'fichier1.cpp', 'fichier2.cpp', 'fichier3.cpp', and 'header1.h'. The main editor area shows the 'CMakeLists.txt' file with the following content:

```
1 cmake_minimum_required(VERSION 2.8)
2 find_package(Imagine)
3 add_executable(programme fichier1.cpp
4   fichier2.cpp fichier3.cpp header1.h)
5 ImagineUseModules(programme Graphics)
```

The 'header1.h' file is also visible, containing:

```
1 #pragma once
2 int f2();
3 int f3();
```

The 'fichier2.cpp' file contains:

```
1 #include "header1.h"
2
3 int f2() {
4   return 2;
5 }
6
```

The 'fichier3.cpp' file contains:

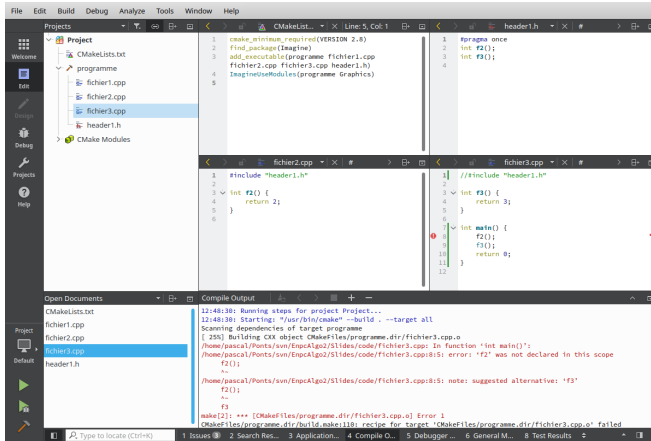
```
1 #include "header1.h"
2
3 int f3() {
4   return 3;
5 }
6
7 int main() {
8   f2();
9   f3();
10  return 0;
11 }
12
```

The 'Compile Output' window at the bottom shows the following logs:

```
12:33:58: Running steps for project Project...
12:33:58: Persisting CMake state...
12:33:51: Starting: "/usr/bin/cmake" --build . --target all
Scanning dependencies of target programme
[ 25%] Building CXX object CMakeFiles/programme.dir/fichier1.cpp.o
[ 50%] Building CXX object CMakeFiles/programme.dir/fichier2.cpp.o
[ 75%] Building CXX object CMakeFiles/programme.dir/fichier3.cpp.o
[100%] Linking CXX executable programme
[100%] Built target programme
12:33:51: The process "/usr/bin/cmake" exited normally.
12:33:51: Elapsed time: 00:01.
```

Pas de  
problème

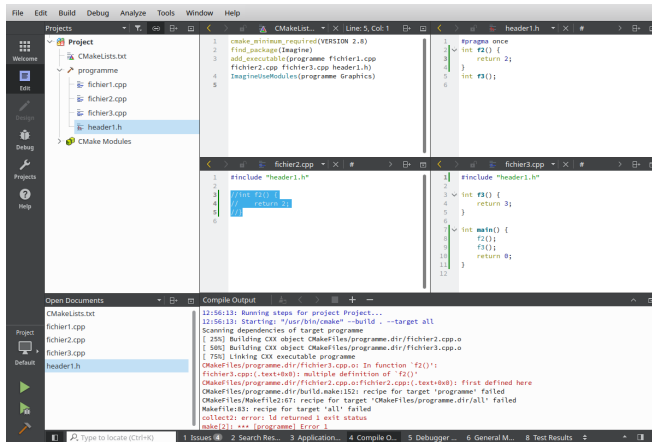
# Problèmes divers



```
File Edit Build Debug Analyze Tools Window Help
Projects
  Project
    CMakeLists.txt
    programme
      fichier1.cpp
      fichier2.cpp
      fichier3.cpp
      header1.h
    CMake Modules
Open Documents
  CMakeLists.txt
  fichier1.cpp
  fichier2.cpp
  fichier3.cpp
  header1.h
Compile Output
12:48:38: Running steps for project Project...
12:48:38: Starting: "/usr/bin/cmake" --build . --target all
Scanning dependencies of target programme
[ 25%] Building CXX object CMakeFiles/programme.dir/fichier3.cpp.o
/home/pascal/Ponts/svn/EnpcAlgo2/Slides/code/fichier3.cpp: In function 'int main()':
/home/pascal/Ponts/svn/EnpcAlgo2/Slides/code/fichier3.cpp:8:5: error: 'f2' was not declared in this scope
f2();
^~
/home/pascal/Ponts/svn/EnpcAlgo2/Slides/code/fichier3.cpp:8:5: note: suggested alternative: 'f3'
f2();
^~
f3
make[2]: *** [CMakeFiles/programme.dir/fichier3.cpp.o] Error 1
CMakeFiles/programme.dir/build.make:118: recipe for target 'CMakeFiles/programme.dir/fichier3.cpp.o' failed
```

Problème  
compilation :  
f2 non  
connue dans  
fichier3.cpp  
(manque  
#include)

# Problèmes divers



The screenshot shows a CMake IDE with the following files in the project:

- CMaakeLists.txt
- programme
- fichier1.cpp
- fichier2.cpp
- fichier3.cpp
- header1.h

The CMakeLists.txt file contains:

```
1 cmake_minimum_required(VERSION 2.8)
2 find_package(Imagine)
3 add_executable(programme fichier1.cpp
4   fichier2.cpp fichier3.cpp header1.h)
5 ImagineUseModules(programme Graphics)
```

The header1.h file contains:

```
1 #pragma once
2 int f2() {
3     return 2;
4 }
5 int f3();
6
```

The fichier2.cpp file contains:

```
1 #include "header1.h"
2
3 //int f2();
4 //return 2;
5
6
```

The fichier3.cpp file contains:

```
1 #include "header1.h"
2
3 int f3() {
4     return 3;
5 }
6
7 int main() {
8     f2();
9     f3();
10    return 0;
11 }
12
```

The Compile Output window shows the following error:

```
12:56:13: Running steps for project Project...
12:56:13: Starting: "/usr/bin/cmake" --build . --target all
Scanning dependencies of target programme
[ 25%] Building CXX object CMakeFiles/programme.dir/fichier2.cpp.o
[ 50%] Building CXX object CMakeFiles/programme.dir/fichier3.cpp.o
[ 75%] Linking CXX executable programme
CMakeFiles/programme.dir/fichier3.cpp.o: In function 'f2()':
fichier3.cpp:(.text+0x8): multiple definition of 'f2()'
CMakeFiles/programme.dir/fichier2.cpp.o:fichier2.cpp:(.text+0x0): first defined here
CMakeFiles/programme.dir/build.make:152: recipe for target 'programme' failed
CMakeFiles/Makefile2:167: recipe for target 'CMakeFiles/programme.dir/all' failed
Makefile:83: recipe for target 'all' failed
collect2: error: ld returned 1 exit status
make[2]: *** [programme] Error 1
```

Problème  
de linker :  
f2 compilée  
deux fois :  
par  
fichier2.cpp  
et  
fichier3.cpp  
⇒ Pas de  
*définition*  
dans header,  
seulement  
dans cpp

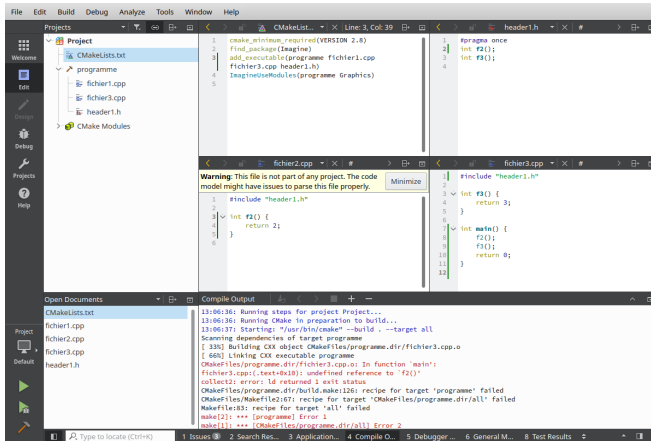
# Problèmes divers

The screenshot shows an IDE with a CMake project. The left sidebar displays the project structure: `Project` (containing `CMakeLists.txt` and `programme`), `programme` (containing `fichier1.cpp`, `fichier2.cpp`, `fichier3.cpp`, and `header1.h`), and `CMake Modules`. The main editor area shows four files: `CMakeLists.txt`, `header1.h`, `fichier2.cpp`, and `fichier3.cpp`. The `Compile Output` panel at the bottom shows the following error:

```
13:01:58: Running steps for project Project...
13:01:58: Starting: "/usr/bin/cmake" --build . --target all
Scanning dependencies of target programme
[ 25%] Building CXX object CMakeFiles/programme.dir/fichier2.cpp.o
[ 50%] Building CXX object CMakeFiles/programme.dir/fichier3.cpp.o
[ 75%] Linking CXX executable programme
/usr/lib/gcc/x86_64-linux-gnu/7/../../../../x86_64-linux-gnu/Scrt1.o: In function '_start':
(.text+0x20): undefined reference to 'main'
collect2: error: ld returned 1 exit status
CMakeFiles/programme.dir/build.make:152: recipe for target 'programme' failed
CMakeFiles/Makefile2:67: recipe for target 'CMakeFiles/programme.dir/all' failed
Makefile:83: recipe for target 'all' failed
make[2]: *** [programme] Error 1
make[1]: *** [CMakeFiles/programme.dir/all] Error 2
```

Problème  
de linker :  
fonction  
main non  
trouvée

# Problèmes divers



The screenshot shows a CMake IDE interface with the following components:

- Projects Panel:** Shows a project named 'Project' with a 'CMakeLists.txt' file and a 'programme' directory containing 'fichier1.cpp', 'fichier3.cpp', and 'header1.h'. It also lists 'CMake Modules'.
- Editor:** Displays four files:
  - CMakeLists.txt:**

```
1 cmake_minimum_required(VERSION 2.8)
2 find_package(Imagine)
3 add_executable(programme fichier1.cpp
4   fichier3.cpp header1.h)
5 ImagineUseModules(programme Graphics)
```
  - header1.h:**

```
1 #pragma once
2 int f2();
3 int f3();
4
```
  - fichier2.cpp:** Shows a warning: "Warning: This file is not part of any project. The code model might have issues to parse this file properly." and contains:

```
1 #include "header1.h"
2
3 int f2() {
4     return 2;
5 }
6
```
  - fichier3.cpp:**

```
1 #include "header1.h"
2
3 int f3() {
4     return 3;
5 }
6
7 int main() {
8     f2();
9     f3();
10    return 0;
11 }
12
```
- Open Documents:** Lists 'CMakeLists.txt', 'fichier1.cpp', 'fichier2.cpp', 'fichier3.cpp', and 'header1.h'.
- Compile Output:** Shows the following log:

```
13:06:36: Running steps for project Project...
13:06:36: Running CMake in preparation to build...
13:06:37: Starting: "/usr/bin/cmake" --build . --target all
Scanning dependencies of target programme
[ 33%] Building CXX object CMakeFiles/programme.dir/fichier3.cpp.o
[ 66%] Linking CXX executable programme
CMakeFiles/programme.dir/fichier3.cpp.o: In function 'main':
fichier3.cpp:(.text+0x10): undefined reference to 'f2()'
collect2: error: ld returned 1 exit status
CMakeFiles/programme.dir/build.make:126: recipe for target 'programme' failed
CMakeFiles/Makefile2:67: recipe for target 'CMakeFiles/programme.dir/all' failed
Makefile:83: recipe for target 'all' failed
make[2]: *** [programme] Error 1
make[1]: *** [CMakeFiles/programme.dir/all] Error 2
```

Problème  
de linker :  
f2 non  
compilée car  
fichier2.cpp  
n'est pas  
dans le  
CMakeLists



# Fonctions

```
type_de_retour nom_fonction()  
type_de_retour nom_fonction([const] type1 [&] arg1)  
type_de_retour nom_fonction([const] type1 [&] arg1, [const]  
                             type2 [&] arg2)
```

- ▶ **nom\_fonction** : combinaison de a–z, A–Z, 0–9 et \_ sauf prefixes 0–9 (interdits), \_\_ et \_[A–Z] (réservés au compilateur)
- ▶ **type\_de\_retour** : **void**, un des types de base (**bool**, (**unsigned**) **char**, (**unsigned**) **short** (**int**), (**unsigned**) **int**, (**unsigned**) **long** (**int**), (**unsigned**) **long long** (**int**), **float**, **double**, **long double**), un objet de la bibliothèque standard (**std::string**, **std::pair<type1,type2>**) ou non (défini par **class** ou **struct**).
- ▶ Passage par référence **&** pour modifier la variable

```
void somme(double& x, double y) { x += y; }  
int main() {  
    double f=3.14, g=-1.0;  
    somme(f,g); //OK, modifie f  
    somme(f,1.0); //OK  
    somme(3.14,g); //Erreur, le premier argument n'est pas une variable  
}
```

# Tableaux

## Statique

```
int m=10;
int tab[m]; //KO, m var
const int n=10;
int tab[n]; //OK, n const
tab[0]=0; //1er element
for(int i=1; i<n; i++)
    tab[i] = tab[i-1]+i;
tab[n] = 0; //BUG indice
```

## Arg de fonction sans réf.

```
void init(int tab[10]) {
    for(int i=0; i<10; i++)
        tab[i]=0; }
int* t1=new int[10]; init(t1);
delete [] t1;
int t2[20]; init(t2);
int t3[5]; init(t3); //KO
```

## Dynamique

```
int m=10;
int* tab=new int[m]; //Alloc
tab[0]=0; //1er element
for(int i=1; i<m; i++)
    tab[i] = tab[i-1]+i;
cout << tab[m-1] << endl;
delete [] tab; //Desalloc
```

## Linéarisation tableau 2D

```
int m=1000, n=500;
float* mat=new float[m*n];
for(int j=0; j<n; j++)
    for(int i=0; i<m; i++)
        mat[j*m+i]=0;
delete [] mat;
```

# Embranchements et boucles

## Embranchements

```
if(condition)
    instructionVrai;
else
    instructionFaux;

if(condition)
    instructionVrai;
else if(condition2)
    instructionVrai2;
else
    instructionFaux;
```

## Boucles

```
for(inst1; cond; inst2){
    inst;
}

while(cond){
    inst;
}

do{
    inst;
}while(cond);
```

```
for ⇔ { //nouveau scope, confine une decl. dans inst1
    inst1;
    while(cond) {
        inst;
        inst2;
    }
}
```

# Classes

- ▶ Alias d'un type existant (les deux deviennent équivalents) :

```
typedef unsigned char byte;
```

- ▶ Nouveau type :

```
struct Matrice {  
    int m, n;  
    double* tab;  
}; //Ne pas oublier le ; apres cette accolade  
void init(Matrice& M) {  
    for(int i=0; i<M.m*M.n; i++)  
        M.tab[i]=0;  
}  
Matrice cree(int nlig, int ncol) {  
    Matrice M = {nlig, ncol, new double[nlig*ncol]};  
    init(M);  
    return M;  
}  
int main() {  
    Matrice M = cree(10,2000);  
    M.tab[2000*3+1416] = 2.718;  
    delete [] M.tab; //Ne pas oublier !  
}
```

# Méthodes

On ajoute la méthode (=fonction de classe) `void Matrice::init()`.

```
class Matrice { //Dans .h
public:
    int m, n;
    double* tab;
    void init() { // def "inline"
        for(int i=0; i<m*n; i++)
            tab[i]=0;
    }
}; //Ne pas oublier le ; apres cette accolade
Matrice cree(int nlig, int ncol) { // dans .cpp
    Matrice M;
    M.m=nlig; M.n=ncol; M.tab=new double[nlig*ncol]};
    M.init();
    return M;
}
```

# Méthodes

Idem avec définition non inline :

```
class Matrice { // dans .h
public:
    int m, n;
    double* tab;
    void init(); //declaration
}; //Ne pas oublier le ; apres cette accolade
Matrice cree(int nlig, int ncol) { // dans .cpp
    Matrice M;
    M.m=nlig; M.n=ncol; M.tab=new double[nlig*ncol]};
    M.init();
    return M;
}
void Matrice::init() { // definition dans .cpp
    for(int i=0; i<m*n; i++)
        tab[i]=0;
}
```

# Opérateurs

- ▶ Les types numériques comprennent les opérateurs suivants :  
+ - \*/ (et % pour types entiers), et les affectations du genre  
+ = ainsi que les comparaisons <, >, <=, >=, ==, !=  
pour les types numériques. Pour les **bool** : ==, !=, !  
(négation unaire, i.e. 1 seul argument), &&, ||.
- ▶ Il faut définir soi-même ceux dont on veut :

```
bool operator==(Matrice m1, Matrice m2) {  
    if (m1.m!=m2.m || m1.n!=m2.n) return false;  
    for (int i=0; i<m1.m*m1.n; i++)  
        if (m1.tab[i]!=m2.tab[i])  
            return false;  
    return true;  
}  
Matrice m1=cree(2,2), m2=cree(2,2);  
cout << m1==m2 << endl; //<=> operator==(m1,m2)  
cout << m1!=m2 << endl; //OK, meme si pas explicite
```

La définition de **operator==** provoque celle de **operator!=**.

# Opérateurs

- ▶ Restriction : les deux types d'un opérateur binaire peuvent différer, mais l'un d'eux doit être un objet.
- ▶ Encapsulation de l'opérateur dans la classe :

```
bool operator==(matrice m1, matrice m2);  
bool Matrice::operator==(matrice m2);
```

- ▶ On peut alors ajouter `operator()` et `operator[]`. `operator()` est spécial car on peut le définir avec un nombre arbitraire d'arguments (y compris 0). `operator[]` n'a qu'un argument.

```
class Matrice { ...  
    double operator()(int i, int j) const;  
    double& operator()(int i, int j);  
};  
cout << M(0,2) << endl; M(1,1) = 0;
```

Notez qu'on a les versions `const` et non-`const` (non appelable pour les matrices non-`const` et permettant de modifier le coefficient).



# Constructeurs/destructeur

- ▶ On cache les champs de Matrice en **private**, seules les méthodes peuvent y accéder. Il est bon d'avoir un ou des constructeurs :

```
class Matrice {  
    double* tab; int m,n;  
public:  
    Matrice(int nligcol); //matrice carree  
    Matrice(int nlig, int ncol);  
    ~Matrice(); // destructeur  
}  
Matrice::Matrice(int nligcol) {  
    m=n=nligcol;  
    tab = new double[m*n];  
}  
Matrice::~~Matrice() {  
    delete [] tab;  
}  
Matrice M(3);
```

- ▶ Si aucun constructeur n'est défini, on peut quand même créer un objet (constr. sans argument implicite). Il faut un constr. sans arg. pour créer un tableau d'objets.
- ▶ On n'appelle jamais explicitement le destructeur, c'est automatique quand l'objet sort de sa portée.

# Constructeurs/destructeur

- ▶ Une fonction (ou méthode) qui prend un objet en arg. sans référence crée une variable locale, construite avec le constructeur par copie :

```
class Matrice {  
public:  
    Matrice(const Matrice& M);  
    void operator=(const Matrice& M);  
}  
Matrice::Matrice(const Matrice& M) {  
    m=M.m; n=M.n;  
    tab = M.tab; // TRES DANGEREUX  
}
```

Le fait de faire `tab=M.tab` provoquera un crash lors de l'appel du destructeur de l'objet copié car on aura déjà libéré son champ `tab` lors du destructeur de la copie. Il faut allouer la propre mémoire pour la matrice (cf TP pour alternative)

- ▶ `operator=` est le plus compliqué : on doit faire l'équivalent du destructeur de l'objet avant de faire la copie de `M`.
- ▶ Inutile de faire de destructeur, const. par copie et `operator=` si votre classe ne fait pas de `new`. Dans le cas contraire, ils sont nécessaires.

# Variables statiques

- Dans une fonction :

```
int my_random() {  
    static bool seeded=false;  
    if(! seeded) {  
        seeded=true;  
        srand((unsigned int)time(0));  
    }  
    return rand();  
}
```

- Dans une classe :

```
struct Tool {  
    static int nb; int serialNumber;  
    Tool() { serialNumber=nb++; }  
};  
int Tool::nb=0; //init, dans cpp  
void func() {  
    Tool a,b;  
    cout << Tool::nb; //OK  
    cout << a.nb << ' ' << b.nb; } //<=> Tool::nb
```

# Divers

- ▶ Utiliser assert (`#include <cassert>`).
- ▶ `std::cin` et `std::cout` (`#include <iostream>`) peuvent aussi être utilisés pour un objet :

```
ostream& operator<<(ostream& str, const Matrice& M) {
    str << M.lignes() << 'x' << M.colonnes();
    return str;
}
istream& operator>>(istream& str, Matrice& M) {
    int nlig, ncol;
    str >> nlig >> ncol;
    M.reset_dim(nlig, ncol);
    return str;
}
cout << M << endl;
cout << "Entrez les nouvelles dimensions de M: -";
cin >> M;
```

- ▶ Le fait de retourner le flot permet d'enchaîner les `<<` et `>>`.
- ▶ Les instructions `continue` et `break` (dans une boucle) permettent de passer à l'itération suivante (en vérifiant la condition) ou de sortir directement.