Documentation et tests en C++

7 mars 2025

Introduction

- Documentation, tests
- Importance dans le développement logiciel en C++
- Le premier lecteur de la doc, c'est vous !
- La première victime des bugs, c'est vous !
- Doxygen : outil de documentation automatique

Exemple de fonction documentée

Listing 1 – Fonction addition documentée avec Doxygen

```
/**
 * \brief Ajoute deux entiers.
 * \param a Premier entier.
 * \param b Deuxième entier.
 * \return La somme de a et b.
 */
int addition(int a, int b) {
    return a + b;
}
```

Génération de documentation

- Doxygen extrait les commentaires pour générer la documentation.
- Commande de base : `doxygen Doxyfile`
- Le fichier `Doxyfile` permet de configurer la documentation.

Exemple de Doxyfile

Listing 2 – Fichier de configuration Doxygen minimal

```
PROJECT_NAME           = "ExempleAddition"
OUTPUT_DIRECTORY       = docs
INPUT                  = .
RECURSIVE              = NO
GENERATE_HTML          = YES
```

Documenter une classe répartie

- Pour une classe répartie sur plusieurs fichiers, documentez principalement le fichier d'en-tête (`.h`).
- Le fichier source (`.cpp`) contient l'implémentation et peut être moins commenté.
- Cette approche centralise la documentation et simplifie la maintenance.

Documentation dans le fichier d'en-tête I

Listing 3 – MaClasse.h - Déclaration et documentation concentrée

```
/**
 * \file MaClasse.h
 * \brief Déclaration de la classe MaClasse.
 */

/**
 * \class MaClasse
 * \brief Exemple de classe documentée uniquement dans
     le header.
 *
 * Cette classe illustre comment centraliser la
     documentation dans le fichier d'en-tête.
 */
class MaClasse {
public:
    /// Constructeur par défaut.
    MaClasse();
```

Documentation dans le fichier d'en-tête II

```
    /// Méthode d'exemple qui effectue une action.
    void faireQuelqueChose();

private:
    int attribut;
};
```

Implémentation minimale dans le fichier source I

Listing 4 – MaClasse.cpp - Implémentation minimale

```
/**
 * \file MaClasse.cpp
 * \brief Implémentation de la classe MaClasse.
 */

#include "MaClasse.h"

MaClasse::MaClasse() : attribut(0) {}

void MaClasse::faireQuelqueChose() {
    // Implémentation de la méthode, la documentation dé
        taillée est dans le .h
}
```

Compilation de la documentation avec Doxygen

Listing 5 – Exemple de Doxyfile pour un projet multi-fichiers

```
PROJECT_NAME        = "ProjetMultiFichiers"
OUTPUT_DIRECTORY    = docs
INPUT               = src include
RECURSIVE           = YES
GENERATE_HTML       = YES
```

- Utilisez l'option `INPUT` pour spécifier tous les répertoires contenant le code source.
- L'option `RECURSIVE` permet d'inclure les fichiers dans les sous-dossiers.
- Exécutez la commande `doxygen Doxyfile` pour générer l'ensemble de la documentation.

Ajout de formules mathématiques dans la documentation

Listing 6 – Exemple d'utilisation de formules mathématiques dans un commentaire Doxygen

```
/**
 * \brief Calcule l'intégrale d'une fonction.
 *
 * La formule mathématique utilisée est :
 * \f[
 *   \int_a^b f(x)\,dx = F(b) - F(a)
 * \f]
 *>
 * Pour une formule en ligne, par exemple : \f$\alpha +
      \beta = \gamma\f$.
 */
double integrale(double a, double b, double (*f)(
    double));
```

Introduction aux techniques de tests

- Pourquoi les tests unitaires ?
- Outils disponibles en C++ : CTest, gtest, Catch2, Boost.Test, CppUnit, etc.
- Introduction à CTest avec CMake

Exemple de fonction à tester

Voici une fonction simple qui calcule le carré d'un entier :

```
int square(int x) {
    return x * x;
}
```

Écriture d'un test unitaire

```
#include <cassert>
#include "square.h"

int main() {
    assert(square(2) == 4);
    assert(square(-3) == 9);
    assert(square(0) == 0);
    return 0;
}
```

Organisation du projet

- src/square.h
- src/square.cpp
- tests/test_square.cpp
- CMakeLists.txt

Fichier CMakeLists.txt

```
cmake_minimum_required(VERSION 3.10)
project(SquareTest)

add_library(square src/square.cpp)
add_executable(test_square tests/test_square.cpp)
target_link_libraries(test_square square)

enable_testing()
add_test(NAME SquareTest COMMAND test_square)
```

Compilation et exécution des tests

```
mkdir build && cd build
cmake ..
make
ctest
```

Analyse des résultats de CTest

- Résultats des tests affichés dans la console
- Logs disponibles en cas d'erreur

Ajout de plusieurs tests dans CTest

```
add_test(NAME TestSquarePositive COMMAND test_square 2
    4)
add_test(NAME TestSquareNegative COMMAND test_square -3
    9)
```

Ajout d'un exécutable principal

```
cmake_minimum_required(VERSION 3.10)
project(SquareProject)

add_library(square src/square.cpp)
add_executable(main_exec src/main.cpp)
target_link_libraries(main_exec square)

enable_testing()
add_executable(test_square tests/test_square.cpp)
target_link_libraries(test_square square)
add_test(NAME SquareTest COMMAND test_square)
```

Exemple de main.cpp

```
#include <iostream>
#include "square.h"

int main() {
    int x = 5;
    std::cout << "Le␣carré␣de␣" << x << "␣est␣" << square
        (x) << std::endl;
    return 0;
}
```

Organisation du projet avec un exécutable principal

- src/main.cpp : Programme principal
- src/square.h, src/square.cpp : Bibliothèque
- tests/test_square.cpp : Tests unitaires
- CMakeLists.txt : Configuration du projet

Compilation et exécution du projet

```
mkdir build && cd build
cmake ..
make
./main_exec # Exécuter le programme principal
ctest       # Lancer les tests
```

Recommandations

- Rédiger des tests simples et isolés
- L'exécution des tests doit être rapide
- Couvrir le plus de cas possibles
- Pour aller plus loin : CI (Continuous Integration) CD (Continuous Deployment), outils gitlab, jenkins, travis, etc.