ECOLE CENTRALE DE NANTES

Option Réalité virtuelle - TP MEDEV
Tests Unitaires & Google Test

10 Décembre 2020

L'objectif de ce TD est d'expérimenter les tests unitaires et d'avoir une première expérience avec **Google Test (GTest)**. Afin d'avoir une base commune pour implémenter les tests, veuillez télécharger ou cloner le code se trouvant à l'adresse suivante : https://github.com/pilllea/TDtestsUnitaires. Plusieurs choix s'offrent à vous pour pouvoir utiliser le framework Google Test. Je vous conseille grandement d'utiliser Visual Studio Community (version 2017 ou plus) dans lequel GTest est déjà inclus.

1 PREMIER TEST À LA MAIN

Veuillez tester la fonction **factorielle** à l'aide d'un opérateur ternaire, i.e., *isValid = (condition) ? instruction si vrai : instruction si faux*. Pensez à tester à la fois des cas ordinaires ainsi que des cas limites (par exemple 0, pour rappel $0! = 1$).
Quelles limites voyez-vous à l'utilisation de ce type de test à la main ?

2 CRÉATION D'UN PROJET GOOGLE TEST SOUS VISUAL STUDIO

Sous Visual Studio (avec la base de projet ouverte) faites Fichier > Ajouter > Nouveau projet... Donnez un nom au projet et valider la configuration. Lancer le fichier de test et observez le résultat.
(A savoir : Pour passer du projet de test au projet de base, dans l'Explorateur de solution, faites un clic droit sur le nom du projet que vous souhaitez lancer, puis sur Définir en tant que projet de démarrage.)

3 Premier test avec Google Test

Commencez par ajouter la ligne `#include "../TDtestsUnitaires/Calculator.cpp"` en haut du fichier "test.cpp". Puis, testez de nouveau la fonction factorielle à l'aide de TEST(TestCaseName , TestName) et des macros `ASSERT_EQ(expected,actual)` et `EXPECT_EQ(expected,actual)`.
Puis désactivez ce test en plaçant un `DISABLE_` dans l'une des entrées de TEST, i.e., `TEST(DISABLE_TestCaseName TestName)` ou `TEST(TestCaseName, DISABLE_TestName)` et observez les résultats.

4 Expérimenter Google Test et le Test Driven Development (TDD)

Comme expliqué lors du cours, le TDD est une méthode agile de code qui est guidée par les tests. Cette dernière est cyclique et comporte les phases suivantes :

1. Écrire un seul test qui décrit une partie du problème à résoudre
2. Vérifier que le test échoue, c'est-à-dire que le code se rapportant à ce test n'existe pas
3. Écrire juste assez de code pour que le test réussisse
4. Vérifier que le test ainsi que ceux pré-existants passent
5. Remanier le code, i.e., améliorer sans altérer le comportement, e.g., enlever les doublons

Tout l'intérêt du TDD consiste à suivre le cycle précédent dans l'ordre. Donc, toute déviation par rapport à l'ordre des questions dans cette feuille réduit l'intérêt de ce travail.

5 Test 1 - EXPECT_DOUBLE

Créez un test à l'aide de la macro TEST(TestCaseName , TestName) pour une fonction nommée "div" qui retourne 2 (type *double*) lorsqu'elle prend entrée les valeurs 4 et 2 (type *double* également). Attention les macros à utiliser pour tester les *float* et *double* sont différentes. Par exemple, il est ici nécessaire d'utiliser `EXPECT_DOUBLE_EQ (expected, actual)`. Créez une fonction vide pour que le code puisse compiler et vérifiez que le test ne passe pas.
L'objectif est de créer le code pas à pas, en utilisant à chaque étape une méthode simple, rapide, et faisant passer les tests existants. Écrivez donc la fonction la plus simple possible permettant de faire passer ce test.

6 Test 2 - EXPECT_NEAR

Pour le second test, réalisez une fonction "div" qui vérifie que le résultat de la fonction "div" lorsqu'elle prend en entrée 7 et 3 retourne 2.333 ± 0.001. La macro correspondante à utiliser est la suivante : `EXPECT_NEAR (expected, actual, absolute_range)`. Vérifiez que le test ne passe pas.
Écrivez la fonction "div" (pour division, bien sûr) la plus simple possible permettant de vérifier le test ci-dessus.

Testez de nouveau la fonction "div" avec la macro `EXPECT_NEAR` pour savoir si le résultat est à 2.333 ± 0.0001 et observez le résultat du test.

7 TEST 3 - ASSERT_EXIT

Pour le troisième et dernier test, utilisez `ASSERT_EXIT(statement, predicate, expected_message)` afin de tester si le programme quitte bien avec le prédicat " : :testing : :ExitedWithCode(-1)" et le message "Error : Division by 0 not possible" lorsque la fonction "div" prend en entrée 7 et 0. Avant de lancer le test ajouter la fonction main suivante à votre fichier de test :

```
"int main(int argc, char **argv) {
::testing::InitGoogleTest(&argc,argv);
return RUN_ALL_TESTS();
}"
```

Vérifiez que le test ne passe pas.
Modifiez votre fonction afin de réussir le test précédent.

8 TEST 4 - A VOUS DE JOUER

Bien que le programme soit simple, d'autres fonctionnalités restent à tester. Identifiez celle primordiale au bon fonctionnement du programme. Modifiez le programme pour la rendre testable. Créez un code pour tester son fonctionnement. Enfin, modifier le programme afin qu'il passe ce test.