Aix*Marseille université

Année 2021-22

Domaine Sciences et Technologies
MASTER INFORMATIQUE

Fiabilité Logicielle : TP 1
Code UE : SINBU22

CUnit : To do

Les environnements de tests unitaires existent pour la plupart des langages et plusieurs peuvent exister pour un même langage (`CPPUnit`, `google test` pour C et C++ par exemple). Dans ce TP vous allez mettre en oeuvre des environnements de tests unitaires pour C. Le prochain TP sera consacré aux tests unitaires avec `JUnit` en Java.

Les modalités de rendus des TP sont décrites dans le document sur le site du cours.

1 Principes généraux

Les environnements de tests unitaires permettent d'écrire des suites de tests selon le schéma suivant :

1. partie initialisation et clôture de la suite de test (test fixture). Par exemple, connexion à une base de donnée, puis déconnexion de celle-ci.
2. mise en place de la suite de test
3. pour chaque test
   (a) Initialisation et clôture pour chaque test,
   (b) calcul des données de test,
   (c) appel de la fonction testée sur les données,
   (d) utilisation d'une assertion pour vérifier que le résultat (value) est égal à la valeur attendue (expected).

L'exécution de la suite de test renvoie un rapport de test indiquant les succès (pass), échecs (fail) des tests ou erreurs (error) déclenchés. Un principe du test unitaire est d'écrire une suite de test pour chaque unité du programme (une classe ou une fonction). Un principe est de ne tester qu'une seule propriété par test : si ce n'est pas le cas le test doit être réécrit en plusieurs tests, donc une seule assertion par test et pas de condition booléenne compliquée (en général).

2 Applications à tester

Cette partie décrit les applications à écrire puis tester.

2.1 Le triangle

Un triangle est déterminé par les longueurs `a`, `b`, `c` de ses 3 cotés. Celle-ci sont mesurées par un dispositif qui remplit un fichier texte ligne par ligne avec une valeur.

L'application comporte deux fonctionnalités :

— Une fonction `Triangle readData(char * filename)` qui renvoie une structure de type triangle dont les cotés correspondent aux trois lignes du fichier.

— Une fonction `int typeTriangle(a, b, c)` qui renvoie le type du triangle dont les cotés sont les réels `a`, `b`, `c`. Ce type est donné par un entier : -1 s'ils ne définissent pas un triangle, 3 si le triangle est équilatéral, 2 s'il est isocèle, 1 s'il est scalène (quelconque).

2.2 Chercher un élément dans un tableau trié

L'application doit chercher un élément de type entier dans un tableau d'entiers triés par ordre croissant.

— La fonction `int* generateSorted(int n)` retourne un tableau d'entiers de taille `n` trié par ordre croissant

— la fonction `int searchInteger(int[] t, int n, int e)` recherche l'élément `e` dans le tableau `t` de taille `n`. Elle renvoie -1 si `e` n'est pas un élément du tableau, sinon elle renvoie l'indice minimal `i` de `t` tel que `t[i]` = `e`. Le coût de la fonction doit être en $O(\log_2(\texttt{n}))$ avec `n` le nombre d'éléments du tableau.

3 `CUnit` : environnement de test pour C

3.1 Présentation rapide

`CUnit` permet de faire des tests en C mais des environnements plus évolués existent pour traiter C++ (et C) comme `CPPUnit` et `google test`. Sous Ubuntu, le package installant `CUnit` fait partie de la distribution, mais on peut aussi l'installer sur son compte indépendamment et l'utiliser comme bibliothèque C.

3.2 Utilisation de `CUnit`

La bibliothèque `CUnit http://www.sourceforge.net/projects/cunit/` n'est pas installée et vous devez donc l'installer sur votre compte (voir la fiche `tp1_getstarted.pdf`). La version à installer est `CUnit-2.1-2` qui n'est pas la dernière dont l'archive n'est pas complète (Voir le fichier `CUnit-Louvain.pdf` provenant de l'université de Louvain pour une documentation sommaire) ainsi que les fichiers monAppli.c et testmonAppli.c disponibles sous AMETICE.

Application Triangle On précise la spécification : la fonction `readData` renverra une structure C dont les 3 champs sont de type `float` correspondant aux valeurs des cotés. Dans le cas de fichiers non conforme ou inexistant, la fonction affectera -1 à chaque champ.

1. Récupérer le fichier triangle.h sous AMETICE pour avoir la définition des profils des fonctions et le type Triangle
2. Programmer les fonctions `typeTriangle` et `readData` dans un fichier `triangle.c` et une fonction main dans un fichier `mainTriangle.c` permettant de l'utiliser.
3. Ecrire les suites de tests `testTypeTriangle.c` et `testReadData.c` et utiliser `CUnit` pour voir si vos fonctions passent les tests. Le site donne une documentation complète mais vous pouvez allez voir un exemple d'utilisation basique ici

   `http://wpollock.com/CPlus/CUnitNotes.htm`
4. Le site AMETICE contient les codes objets de trois réalisations[1] de `triangle.c` : `triangle1.o`, `triangle2.o`, `triangle3.o`. Les tester et donner les résultats obtenus.

Recherche d'un élément dans un tableau trié Réaliser une suite de tests les deux fonctions liées à la recherche d'un élément dans un tableau trié décrite ci-dessus. Utiliser votre suite de tests sur l'implémentation C que vous avez réalisée.

4 Travail à rendre

Voir le cours AMETICE pour la spécification des rendus (forme et date).
En plus des fichiers obligatoires, l'archive zip contiendra deux répertoires :

1. Le répertoire `triangle` qui contient tous les fichiers sources et de test pour l'appli- cation `Triangle` en C.
2. Le répertoire `search` qui contient tous les fichiers sources et de test pour la rechercche d'un élément dans un tableau trié.
3. Le compte-rendu contiendra toutes les informations utiles décrivant vos choix pour le codage et la réalisation des tests

1. Ces fichiers `obj` ont été compilés sur la machine virtuelle LUMINUX-2201.