Faculté des Sciences
Aix*Marseille Université

Master Informatique 1
Fiabilité Logicielle

TP Tests Unitaires en C et Python

Département Informatique et Interactions
Année 2018-2019

---

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 et python. Le prochain TP sera consacrés aux tests unitaires en Java avec `JUnit` . **La section 5 précise les consignes pour les rendus de TPs.** L'intitulé du cours sous AMETICE est
`[18]-S2-Fiabilité Logicielle (Denis Lugiez)`.

# 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 connection à une base de donnée, puis deconnection de celle-ci.
2. mise en place de la suite de test
3. pour chaque test
   (a) Initialisation et clotûre 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 indicant les échecs (fail), succès (pass) des tests ou erreurs (error) déclenchées.

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 compliqué.

# 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 fonctionalités :

— Une fonction `readData(nomDeFichier)` qui renvoie un tableau de 3 réels qui correspondent aux trois lignes du fichier.

— Une fonction `typeTriangle(a, b, c)` qui renvoie le type du triangle dont les cotés sont les réels `a,b,c` : $-1$ 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**. Le type `Tableau` a une fonctionnalité `getDim` qui renvoie la taille du tableau et `getValues` qui renvoie le tableau des valeurs.

`chercherElt(int elt, Tableau tab)` renvoie `-1` si `elt` n'est pas un élément du tableau, sinon elle renvoie un indice $i$ du tableau tel que `tab[i] = elt`.

# 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 googletest. 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 Partie `CUnit` du TP

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 `utilisationCUnit.pdf` sur le site AMETICE. 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 utilisation basique ainsi que les fichiers `essai.c` et `testEssai.c` disponibles sous AMETICE.

### 3.2.1 Application Triangle

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

1. Programmer les fonctions `typeTriangle` et `readData` dans un fichier `triangle.c` et une fonction `main` dans un fichier `mainTriangle.c` permettant de l'utiliser.
2. 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`.

### 3.2.2 Application : test de la recherche d'un élément

Ecrire le type `Tableau` et ses fonctionalités (utiliser une structure).

1. Ecrire une suite de test `CUnit` pour la fonction `chercherElt`.
2. Programmer la fonction `chercherElt` et la vérifier avec les tests.
3. Récupérer les fonctions `chercherElt1` et `chercherElt2` sur le site et les tester. Conclusion ?

# 4 `unittest` : environnement de test en python

## 4.1 Présentation rapide

Un équivalent de `CUnit` pour python est le package `unittest` . Pour vous rappeler les bases de python, vous pouvez aller ici : `https://docs.python.org/fr/3.5/tutorial/`. Pour écrire une suite de test on crée une classe de test qui contient une suite de tests. Par exemple :

```
class EssaitTest(unittest.TestCase):
    #un test : le nom est de la forme test... ici testGetVal
    def testGetVal(self):    #un test
        e = Essai(1.0)
        val = e.getVal()
```

```
        expected = 1.0
        self.assertEquals(expected, val)

if __name__ == '__main__':  # ce qu'il faut ajouter pour que les
    unittest.main()  # tests soient effectués sous l'interpréteur
```

Les deux dernières lignes ne font pas partie de la classe mais sont nécessaires pour l'exécuter dans l'interpréteur (facultatives sous certains IDE).

### 4.2 Partie `unittest` du TP

Reprendre la partie `CUnit` et la reprogrammer en python et en utilisant `unittest` aussi bien l'application `Triangle` que l'application `Chercher un élément`. Le type `Tableau` pour la recherche d'un élément sera remplacé par le type `list` natif en python avec la fonction `len` qui permet de connaitre la longueur d'une liste. La fonction de recherche fera partie d'un classe `Chercher` qui a un attribut `tabElement` qui est une liste d'entiers et le profil de la fonction sera `ChercherElement(self, elt)`. La classe de test s'appelera `testChercher`.

## 5 Rendu des TPs

### 5.1 Modalités

`A respecter absolument`

- Chaque groupe de TP rendra une archive au format zip appelée `TPTEST-GROUPEi.zip` (avec i le numéro du groupe) à déposer dans l'activité `Devoir Rendu TP TEST` du site AMETICE **à la fin du TP**. **Les documents pdf seront écrits en respectant le format de documents donné dans la partie Biblio du cours AMETICE**.
- La décompression de l'archive créera un répertoire TPTEST-GROUPEi
- Chaque fichier source contiendra une `en-tete` qui est un commentaire avec les noms des membres du groupe et tout autre information jugée utile.
  Exemple minimaliste d'en-tête (en C ou Java) :

```
/**
*@author : Etu1
*@author : Etu2
*
*/
```

**ATTENTION** : rendre uniquement une partie bien faite est bien mieux évalué que tout le TP baclé et mal fait.

## 5.2 Travail demandé

L'archive zip contiendra deux projets :

1. Le projet `triangle` qui contient tous les fichiers sources et de test pour l'application Triangle **en C**. Un fichier `CR.pdf` contiendra toutes les informations utiles décrivant vos choix pour le codage et la réalisation des tests.
2. Le projet `recherche` qui contient l'application de recherche et les classes de test **en python**.

## 5.3 Date de rendu

Par défaut, tous les TP (celui-ci et les suivants) sont à rendre **le mercredi suivant le TP avant 23h59** dans l'activité AMETICE devoir correspondant à votre site et votre parcours. Les rendus par email ou ne correspondant pas au format ne seront pas pris en compte.