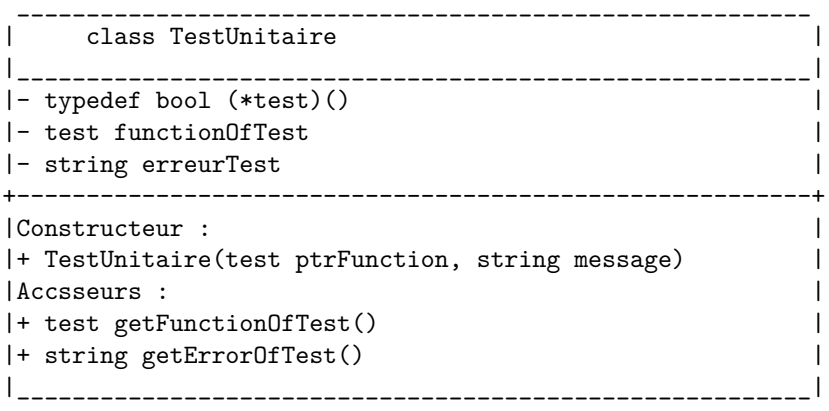
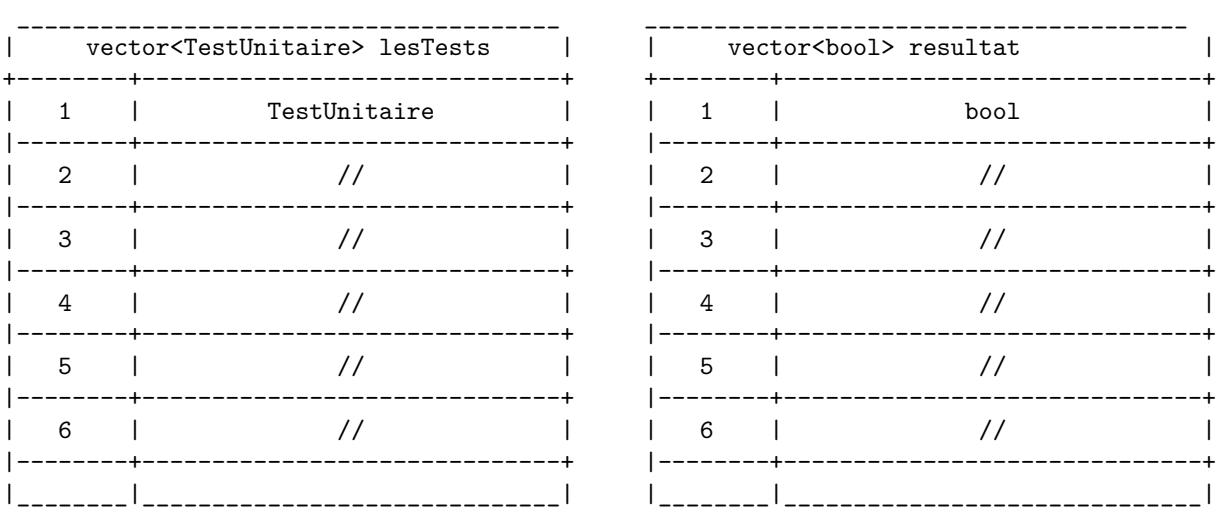
TP 1
TAD Couleur

Algorithmique et Structure de données
Semestre 3

1 Schéma du TAD Test





2 A quoi sert le classe *TestUnitaire* ?

La classe *TestUnitaire* fournit des outils pour tester chaque fonction et repérer l'erreur correspondante. La classe associe un pointeur sur la fonction défaillante et un message d'erreur approprié.

- *functionOfTest* désigne un pointeur vers la fonction qui pose problème (accesseur : *getFunctionOfTest()*)
- *errorTest* contient le message de l'erreur à afficher (accesseur : *getErrorTest()*)

3 Comment fonctionne la procédure *runAllTests* ?

La procédure *runAllTest* initialise les *TestUnitaires*, les effectuent et nous signale une erreur le cas échéant.
Elle initialise le vecteur *lesTest* avec la fonction *initializeAllTests* et rempli le vecteur résultat de 'false'.
Ensuite, pour chaque *TestUnitaire*, ont vérifie que les fonctions ne posent pas de problème et modifie le vecteur résultat en conséquence.

- true ⇒ pas de soucis
- false ⇒ erreur à traiter

Elle vérifie que tous les tests ont réussis (*testsAllOkay(resultat)*)
Si les tests ont réussi on affiche "Tous les tests sont Okay", dans le cas contraire on affiche le message d'erreur correspondant aux différentes erreurs rencontrées à l'aide de la fonction *getErrorTest()*

4 Comment ajouter un nouveau test ?

1. Ajouter un fonction de test dans le main.cpp
2. Ajouter une entrée dans cette fonction :

```
1 void initializeAllTests(vector<TestUnitaire>& lesTests) {
2   // ... Tests précédents
3   lesTests.push_back(TestUnitaire(&nomDeMaFonctionDeTest,
4     string ("Message message d'erreur pour déterminer le problème"));
5 }
```

Listing 1 – Initialisation du test

3. Recompiler et tester sa fonction de test en faisant une erreur volontaire et vérifier que le test échoue, puis corriger l'erreur et vérifier que le test réussi.
4. En cas d'erreur non volontaire, corriger cette erreur dans notre fonction de test.

5 Que garder pour écrire les tests d'une nouvelle classe ?

On garde le TAD étudié en 1. : les deux vecteurs (*vector<bool> resultat* et *vector<TestUnitaire> lesTests*). De même pour la classe *TestUnitaire*.

```
1  vector<bool> resultat;
2  vector<TestUnitaire> lesTests;
3  class TestUnitaire{
4     private :
5        // déclaration d'un pointeur de fonction
6        // la fonction ne prend pas de paramètres et renvoie un booléen
7        typedef bool (*test)();
8        test functionOfTest;
9        string errorTest;
10
11    public :
12       // constructeur de test
13       TestUnitaire(test ptrFunction, string message)
14       {
15          this->functionOfTest = ptrFunction;
16          this->errorTest = message;
17       }
18       // les accesseurs en lecture
19       test getFunctionOfTest() const { return this->functionOfTest; }
20       string getErrorTest() const { return this->errorTest; }
21 };
```

Listing 2 – Classe TestUnitaire vector resultat et vector lesTests

Puis toutes les fonctions d'automatisation des tests :

```
1 void initializeAllTests(vector<TestUnitaire>& lesTests)
2 bool testsAllOkay(vector<bool>& resultats)
3 void runAllTests()
4 int main(int argc, char *argv[])
```

Listing 3 – Prototype des fonctions à garder

Pour ajouter un test, cf. Question 4