LOG1000 – Ingénierie Logicielle – TP4
Tests Unitaires

Objectifs:

- Faire des tests unitaires et garantir la qualité de votre logiciel.
- S'initier aux outils de tests unitaires.

Les outils :

Vous allez utiliser l'outil de tests « CppUnit » pour tester les fichiers sources fournis avec le TP.

Durant la première partie du TP, vous serez ramené à tester les fonctionnalités de la classe « Vector » et de la classe « Stack», qui sont définies dans les fichiers « Vector.h », « Stack.h » et implémentées dans « Vector.cpp », « Stack.cpp ». Le fichier « main.cpp » présente un exemple simple d'utilisation du Vector et du Stack.

En principe, Vector et Stack représentent des structures qui permettent de stocker des éléments d'un type donné. Dans le cadre de ce laboratoire, on considère juste les chaines de caractères (string) comme vecteurs.

Veuillez utiliser le fichier « Makefile » pour compiler le code fourni dans ce TP, comme vous l'avez vu dans le TP1.

Enoncé :

Dans le cadre de ce TP, on vous demande de faire des tests unitaires pour les méthodes principales de la classe « Vector » et la classe « Stack» dans le but d'assurer leur bon fonctionnement. Afin de réaliser de bons tests unitaires, il faut créer un test convenable pour chaque chemin indépendent (et possible) d'une méthode à tester, ce qui revient à élaborer en première partie un diagramme de flot de contrôle (Control Flow Graph).

E1) Diagramme de flot de contrôle [/60]

Afin de couvrir les méthodes implémentées avec les tests unitaires de manière optimale, vous devez d'abord faire un diagramme de flot de contrôle pour chacune des méthodes « insert », « remove », « get » et « set ».

À partir du code source (voir “Vector.cpp” et “Stack.cpp”), dessinez les diagrammes de flot de contrôle pour les 3 méthodes « insert », « get » et « set » de la classe Vector [/15], et pour les 3 méthodes «pop», « top » et «push» de la classe Stack [/15]

1. Pour chaque diagramme, calculez la complexité cyclomatique. N’oubliez pas de calculer la complexité avec les deux approches vues en classe pour contrôler vos résultats. [/10]
2. Sur chacun des diagrammes, tracez les chemins nécessaires à parcourir pour couvrir toutes les conditions. [/20]

Vous pouvez faire des dessins avec Powerpoint ou un autre logiciel (par exemple UMLet), puis inclure les diagrammes résultants dans votre rapport. Alternativement, vous pourriez aussi utiliser un format textuel pour les chemins. Le plus important est que votre réponse ne contient pas d’ambiguïtés.

E2) Cas de tests et jeu de données [/20]

Afin d’appliquer vos tests unitaires, vous devez d’abord établir les cas de tests et bâtir un jeu de données. Cette phase n’est que de la conception, aucune implémentation ne doit être faite avec CppUnit à ce point (ça viendra dans E3).

Dans le cadre des méthodes à tester, veuillez définir pour chaque chemin des entrées (des valeurs pour les paramètres de la méthode à tester ou d’autres objets ou variables qui peuvent être manipulés) et la sortie attendue. Par exemple, si on veut tester la méthode suivante :

```
int foo (int x ) {
    if (x > 10) {
        return 1;
    } else {
        return 0;
    }
}
```

Des entrées et des sorties attendues seraient par exemple :

Entrées	Sorties
20	1
5	0

Utilisez une notation similaire pour spécifier les jeux de tests correspondant aux chemins identifiés dans E1. Si un certain chemin n’est pas faisable en pratique, mentionnez-le.

E3) Implémentation et exécution des tests unitaires [/40]

Dans cette section, il faut implémenter les tests unitaires qui correspondent aux chemins que vous avez identifiés dans « E1 », et les jeux de tests et les résultats attendus que vous avez énumérés dans « E2 ». Pour cela, il faut compléter le squelette fourni dans le dossier « tests ». Il contient un fichier source par méthode testée, par exemple le fichier « getTest.h » est utilisé pour tester la méthode « get » de la classe « Vector ». Pour plus d'information sur CppUnit, on vous réfère vers http://www.yolinux.com/TUTORIALS/CppUnit.html.

1. Implémentez vos cas de tests avec CppUnit, en ajoutant une méthode par chemin dans le fichier source approprié. Par exemple, il faut ajouter la méthode « test3 » dans le fichier « removeTest.h » pour tester le 3ème chemin que vous avez identifié dans E1 et E2 pour la méthode « remove » de la classe « Vector ». Il est permis d'ajouter des nouvelles méthodes, mais il ne faut pas changer la signature des classes fournies (leurs noms). Attention, votre code source sera évalué dans cette question, assurez-vous d'écrire un code de bon qualité et bien documenté. [/30]

2. Pour exécuter vos tests, il faut exécuter la commande « make » et ensuite lancer l'exécutable « testsVector ». Exécutez les tests unitaires et copiez les résultats affichés dans votre rapport. [/10]

Contribution au projet Ring [/30]:

Dans la deuxième partie du TP, vous allez contribuer à l'implémentation des tests unitaires du projet open source « Ring » (https://github.com/savoirfairelinux/ring-daemon.git). Notez qu'il ne faut pas compiler ni exécuter ces tests, seulement les concevoir et implémenter en CppUnit.

il faut que vous choisissiez deux méthodes du tableau ci-dessous :

Fichier	Méthodes
archiver.cpp	std::vector<uint8_t> decompress(const std::vector<uint8_t > &str);
Utif8_utils.cpp	bool utf8_validate_c_str(const char *str, sszie_t max_len, const char **end);
Utif8_utils.cpp	bool utf8_make_valid(const std::string & name);
Fileutils.cpp	std :: vector <uint8_t>loadFile(const std::string& path); int removeAll(const std :: string& path) ;
Pattern.cpp	std :: string Pattern ::group(const char* groupeName);
Base64.cpp	uint8_t* ring_base64_decode(const char* input, size_t input_length, uint8_t* output, size_t* output_lentgh);
Base64.c	uint8_t* ring_base64_encode(const char* input, size_t input_length, uint8_t* output, size_t* output_lentgh);
Ringaccount.cpp	static const std::string stripPrefix(const std::string& toUrl);
Preferences.cpp	void verifyAccountOrder(const std:: vector<std::string> &accountIDs);

Puis:

1. À partir du code source, dessinez les diagrammes de flot de contrôle pour les méthodes choisies. [/10]
2. Calculez la complexité cyclomatique avec les deux approches vues en classe pour contrôler vos résultats. [/10]
3. Tracez les chemins nécessaires à parcourir pour couvrir toutes les conditions. [/10].

Considérations importantes pour la fin du TP:

Toujours faire un « git add » des nouveaux fichiers, un « git commit » et un « git push » de vos dernières modifications pour que l'on puisse voir la dernière version de votre travail lors de la correction. Si vous ne faites pas de commit, il se peut que l'on évalue une version différente de votre TP local sur le serveur Git.

Vérification que vos travaux sont présents dans le répertoire Git du serveur:

Vous pouvez utiliser la commande « git ls-files » afin de voir les fichiers dans le dernier snapshot de l'entrepôt Git lui-même. Remplacez XX par le numéro de votre équipe. Ce que vous verrez dans cette liste correspond à ce que l'on verra pour la correction.

!! DATE LIMITE DE REMISE !!

Lundi 3 Avril 2017 avant 23h55.

Tout ce qui est «commit» après cette date ne sera pas considéré dans la correction. !!

Pénalités pour retard: 10% par jour !!