

## TD n°1 - Tests de couverture en langage C

**Résumé** : Cette feuille compare les différentes techniques de test structurel et fonctionnel, ainsi que les aspects du test en boîte noire/boîte blanche, dans le cadre purement impératif du langage C.



Ajouter le répertoire `/net/ens/renault/local/bin` dans votre variable `PATH` avant de commencer le TD, et vérifier alors que la commande `gcc --version` renvoie une valeur supérieure ou égale à 4.1.

### Exercice 1: Pour se chauffer ...

Considérons la fonction C suivante dont la spécification annonce qu'elle permet de calculer le `ppcm` (plus petit commun multiple) de deux nombres entiers donnés :

```
1 int ppcm(int x, int y) {  
2     int p, mincm;  
3  
4     p = x*y;  
5     while ((p > x) && (p > y)) {  
6         if ((p%x==0) && (p%y==0)) {  
7             mincm = p;  
8         }  
9         p = p-1;  
10    }  
11  
12    return mincm;  
13 }
```

Exemples :

- `ppcm(2,3) → 6`
- `ppcm(3,3) → 3`

*Remarque* : Cet exercice applique les techniques de test structurel du code afin de pouvoir déterminer si cette fonction est correcte ou non. Faire attention **à ne pas modifier le code avant** que l'exercice ne vous le demande.

1. Écrire un programme C qui prend en paramètre deux entiers et qui affiche le résultat de la fonction `ppcm`, et cela *sans* modifier le fichier `ppcm.c`.
2. Modifier ce programme C de manière à prendre en paramètre un jeu de test de manière à ce qu'il affiche `PASSED` ou `FAILED` selon que le test soit réussi ou non (et cela sans utiliser les assertions de la bibliothèque `<assert.h>`)  
Quels problèmes peuvent poser les `assert` dans l'optique de faire du test structurel ?
3. Construire le graphe de flot de contrôle de ce code à la main.

## Exercice 2: Test structurel / Couverture des noeuds

L'option de compilation `-fdump-tree-cfg` de `gcc` permet d'obtenir un fichier `.cfg` contenant une représentation du graphe de flot de contrôle du code.

1. Comparer votre graphe avec le fichier généré par `gcc` en compilant le fichier `ppcm.c` avec cette option.

▷ Le programme `crystal` permet de générer facilement des graphes de flot de contrôle de fichiers source C au format `dot`. Il peut être téléchargé à l'adresse suivante : <http://www.cs.cornell.edu/projects/crystal/>.  
Le programme `dot` permet alors de convertir le fichier obtenu au format Post-script.

2. Construire le graphe de flot de contrôle en utilisant `crystal` pour générer un fichier `.dot` et les outils de `graphviz` pour le transformer en `.ps` :

```
crystal ppcm.c -emit-dot && dot -Tps -o ppcm.ps ppcm.dot
```

A partir de maintenant, nous allons utiliser le compilateur `gcc` pour faire des tests de couverture des noeuds.

3. Compiler le programme avec les options de compilation suivantes : `--coverage` (qui implique `-ftest-coverage` et `-fprofile-arcs`). Regarder la page de manuel de `gcc` pour voir l'intérêt de ces options.

▷ Le programme `gcov` fourni avec `gcc` est un programme de calcul de couverture. Lorsqu'un code est compilé avec l'option `--coverage`, l'exécution de ce code génère deux fichiers `.gcno` et `.gda` contenant les résultats de toutes les exécutions de ce code. L'utilitaire `gcov` permet de mettre en forme ces résultats dans un fichier `.gcov`.

4. Lancer le jeu de test (1,1,1) puis `gcov` sur `ppcm.c` pour calculer le taux de couverture.

```
gcov ppcm.c
```

Ceci génère un fichier `ppcm.c.gcov` que l'on peut visualiser, ainsi qu'un calcul de pourcentage donnant le taux de couverture des noeuds.

5. Compléter le jeu de test précédent avec (2,3,6) et vérifier que vous obtenez une couverture « tous les noeuds » de 100%.

### Exercice 3: Test structurel / Couvertures plus complètes

Manifestement, le code ne fonctionne pas lorsque les valeurs données en entrée sont égales. On propose de corriger l'erreur en rajoutant les deux lignes suivantes :

```
1 int ppcm(int x, int y) {
2     int p, mincm;
3
4 +   if (x == y)
5 +     return x;
6 +
7     p = x*y;
8     while ((p > x) && (p > y)) {
9         if ((p%x==0) && (p%y==0)) {
```

1. Supprimer les fichiers générés par gcov, et relancer le jeu de test précédent. S'assurer que l'on a toujours une couverture des noeuds de 100%. A votre avis, le programme est-il correct pour autant ?

▷ L'outil **gcov** permet de calculer la couverture des branches lorsqu'il est exécuté avec l'option **-b**. Le fichier **.gcov** généré contient des informations permettant de savoir quelles branches n'ont jamais été exécutées.



Seule la ligne "taken at least once" compte la couverture des branches.

2. Calculer le critère de couvertures des branches de **ppcm.c**. Trouver un jeu de test pour atteindre une couverture des branches de 100%

Il existe toujours un problème avec ce code, qui apparaît lorsque le PPCM est égal à l'une des valeurs passée en paramètre. Remarquer que selon le jeu de test, ce problème peut être détecté ou pas par une analyse structurelle.

3. Proposer une correction du bug et l'appliquer dans le code.

Nous avons maintenant un jeu de test qui permet de couvrir les noeuds et les branches du code. Cela pourrait laisser à penser que le code a été suffisamment bien testé pour qu'il soit correct.

4. Reconstruire le graphe de flot de contrôle du code, et calculer le critère « all-path coverage ». Pour chaque chemin, exhiber un jeu de test qui passe par ce chemin.
5. A l'aide de ces jeux de test, trouver le bug restant dans le code, et le corriger.

#### Exercice 4: Test fonctionnel vs. test structurel

Le code suivant permet de calculer le produit des deux paramètres  $a$  et  $b$ , sans faire appel directement à d'autres multiplications que des multiplications et des divisions par 2 (méthode dite "du paysan russe") :

```
1 int mul(int a, int b) {  
2     int m;  
3  
4     while (b > 0) {  
5         if (b%2 == 1)  
6             m = m-a;  
7         a = a*2;  
8         b = b/2;  
9     }  
10    return m;  
11 }
```

Exemples :

- mul(2,3) → 6
- mul(3,3) → 9

1. Générer le graphe de flot de contrôle pour cette fonction.
2. Appliquer les méthodes de test structurel vues précédemment sur cette fonction à l'aide de **gcov**. En particulier, construire un jeu de test permettant d'obtenir une couverture des noeuds et des branches de 100%.

▷ Le programme **lcov** (disponible à l'adresse <http://ltp.sourceforge.net/coverage/lcov.php>) permet de générer une sortie HTML plus attrayante que **gcov** pour visualiser les couvertures obtenues.

3. À l'aide des commandes suivantes, générer un ensemble de pages HTML permettant de visualiser la couverture des noeuds et des branches de vos jeux de test :

```
1 # Initialize lcov  
2 lcov --directory . --zerocounters;  
3 # Execute some tests  
4 (...)  
5 # Generate HTML report  
6 lcov --directory ./ --capture --output-file test.info;  
7 genhtml -o html test.info
```

Dans le cas de la fonction **mul**, il est parfaitement possible d'obtenir des tests de couverture de 100% sans que la fonction ne soit entièrement correcte. Il devient alors nécessaire d'appliquer des méthodes de test fonctionnel.

4. Quelles sont les classes d'équivalence (pour le test fonctionnel) sur les paramètres de la fonction **mul** ?
5. Compléter le jeu de test précédent et corriger entièrement le code de la fonction **mul** afin qu'elle fonctionne correctement sur toute entrée entière.

### Exercice 5: Test boîte noire

Pour chacune des fonctions suivantes, récupérer le fichier `.o` correspondant. Ensuite, proposez un jeu de test suffisamment exhaustif pour permettre de vérifier si le fichier objet fourni vérifie cette spécification ou non. Indiquer ensuite pour chaque test s'il est passé ou pas.

1. Fonction majuscule : (Fichier : `up.o`)

**`char* uppercase(char* source)`**

“A partir de la chaîne de caractères *source*, renvoie une chaîne de caractères dans laquelle on a mis en majuscule la première lettre d'un mot situé en début de chaîne ou de tout mot situé après un caractère d'espace.”

*Ex.* : `uppercase("alpha_et_omega")` renvoie la chaîne `"Alpha_Et_Omega"`

2. Fonction de translation : (Fichier : `tr.o`)

**`char* translate(char* from, char* to, char* source)`**

“A partir de la chaîne de caractères *source*, renvoie une chaîne de caractères dans laquelle on a remplacé tout caractère de *s* par le caractère de même position dans *to*.”

*Ex.* : `translate("op","as","coleoptere")` renvoie la chaîne `"caleastere"`

3. En conclusion, lesquels de ces deux fichiers fournissent-ils effectivement une fonction correcte par rapport à sa spécification ?

- ▷ À partir de cet énoncé de TD, il est demandé de fournir :
- une archive des sources réalisées en `.tar.gz` ;
  - un fichier PDF de compte-rendu.

Le compte-rendu est censé être écrit sans fioritures, dans un français convenable, avec un style qui montre que vous comprenez la démarche présentée dans les exercices, avec ses qualités mais aussi ses limites (éviter à tout prix le style "question/réponse").